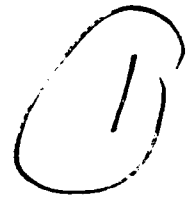


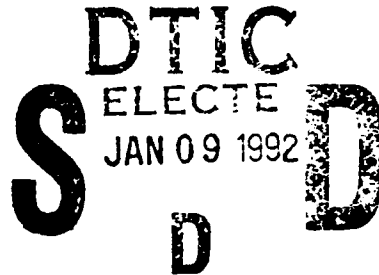
**AD-A244 984**



**SOFTWARE DESIGN DOCUMENT  
GT Real-Time Software Host CSCI (9B)**

Volume 2 of 2 Sections 2.12.20 - 3.2 and Appendices

**June, 1991**



**Prepared by:**

BBN Systems and Technologies,  
A Division of Bolt Beranek and Newman Inc.  
10 Moulton Street  
Cambridge, MA 02138  
(617) 873-3000 FAX: (617) 873-4315

**Prepared for:**

Defense Advanced Research Projects Agency (DARPA)  
Information and Science Technology Office  
1400 Wilson Blvd., Arlington, VA 22209-2308  
(202) 694-8232, AUTOVON 224-8232

Program Manager for Training Devices (PM TRADE)  
12350 Research Parkway  
Orlando, FL 32826-3276  
(407) 380-4518

**92-00256**



**92 1 0 062 1**

# REPORT DOCUMENTATION PAGE

Form Approved  
OPM No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of Management and Budget, Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE  June 1991	3. REPORT TYPE AND DATES COVERED  Software Design Document	
4. TITLE AND SUBTITLE  Software Design Document GT Real-Time Software Host CSCI (9B)			5. FUNDING NUMBERS  Contract Numbers: MDA972-89-C-0060 MDA972-89-C-0061	
6. AUTHOR(S)  Author not specified.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Bolt Beranek and Newman, Inc. (BBN) Systems and Technologies; Advanced Simulation 10 Moulton Street Cambridge, MA 02138			8. PERFORMING ORGANIZATION REPORT NUMBER  Advanced Simulation #: 9117	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)  Defense Advanced Research Projects Agency (DARPA) 3701 North Fairfax Drive Arlington, VA 22203-1714			10. SPONSORING/MONITORING AGENCY REPORT NUMBER  DARPA Report Number: None.	
11. SUPPLEMENTARY NOTES  None				
12a. DISTRIBUTION/AVAILABILITY STATEMENT  Distribution Statement A: Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE  Distribution Code: A	
13. ABSTRACT (Maximum 200 words)  A Simulation Network (SIMNET) project Software Design Document that describes the GT Real-Time Software Host Computer Software Configuration Item (CSCI number 9B) of the SIMNET hardware and software training system for vehicle crew training and operational training.				
14. SUBJECT TERMS  SIMNET Software Design Document for the GT Real-Time Software Host CSCI (CSCI 9B).			15. NUMBER OF PAGES	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Same as report.	

**2.12.20 show\_effect\_msg.c**

The `show_effect_msg` function adds new special effects to the active effects table in active area memory. This function is called whenever the Simulation Host sends a `MSG_SHOW_EFFECT` message. If the system contains two backends, `show_effect_msg` is called twice (each active area memory has its own active effects table).

The function call is `show_effect_msg(msgp, pdbase, vme_offset)`, where:

*msgp* is a pointer to the `MSG_SHOW_EFFECT` message

*pdbase* is a pointer to the primary database block

*vme\_offset* is the VME address offset to AAM

`show_effect_msg` finds the load module in which the effect is to be placed, then adds the effect to the multiple-frame effects list.

Called By: `msg_show_effect`

Routines Called: `FIND_LM`

Parameters: `MSG_SHOW_EFFECT` `*msgp`  
`DB_INFO` `*pdbase`  
`INT_4` `vme_offset`

Returns: `none`

Accession For	
NTIS	CRA&I
DTIC	Lab
Unannounced	
Justification	
By	
Distribution	
Availability	
Dist	Availability Special
A-1	



### 2.13 Real-Time Processing (/cig/libsrc/librtt)

The Real-Time Processing CSC is responsible for setting up and running the simulation using messages sent from the Simulation Host. The primary tasks performed by the functions in this CSC are the following:

#### **System configuration**

- Parses the command line entered by the user to initialize the system.
- Configures Ballistics.
- Opens the terrain database and Dynamic Elements Database (DED) files.
- Configures the color table.
- Configures the database.
- Downloads trajectory tables to Ballistics.
- Downloads textures data to the ESIFA.
- Initiates viewport configuration and 2-D overlay generation.

#### **Simulation processing**

- Processes all runtime messages sent by Simulation Host.
- Performs matrix concatenations to adjust the viewport displays as the simulated vehicle moves and rotates.
- Performs model control on other vehicles in the simulation environment.

#### **Ballistics interface**

- Notifies Ballistics of state changes and changes to simulation parameters (e.g., addition of static vehicles).
- Forwards requests from the Simulation Host to process rounds and trajectories, obtain terrain feedback data, and enable AGL processing.
- Processes messages returned from Ballistics.

#### **Database management**

- Loads the correct portion of the terrain database into active area memory.
- Loads new rows or columns as necessary during the simulation, to keep the simulated vehicle in the center of active area memory.

#### **Database feedback**

- Provides information to the Simulation Host on the terrain surrounding the simulated vehicle.

#### **Hardware tests (not currently implemented)**

- Runs hardware tests requested by the Simulation Host.

#### **File control**

- Transfers files to and from the CIG based on Simulation Host requests.

#### **Miscellaneous**

- Provides functions for interfacing to the Force board.
- Provides functions for interfacing with the EVC (Ethernet VME Controller).
- Allows the Gossip user to display calibration images.
- Provides a mechanism for recording message packets to a file.
- Manages cloud models.
- Builds M1 and M2 gun overlays for viewports connected to T backends.



At startup, various functions are called to initialize active area memory, verify that the communications interface to the Simulation Host is functional, and start Ballistics. The upstart function then processes the messages sent by the Simulation Host to place the CIG in a specified state. The CIG states that can be set are:

<b>Database setup</b>	Prepares the CIG to run a simulation. If this state is requested, upstart passes control to db_mcc_setup.
<b>File control</b>	Used to transfer files to and from the CIG. If this state is requested, upstart passes control to file_control.
<b>Test mode</b>	Used to run hardware tests. If this state is requested, upstart passes control to hw_test.
<b>MCC setup</b>	Prepares the CIG to act as an MCC station. This mode is not currently used.

If database setup is specified, db\_mcc\_setup processes messages from the Simulation Host to configure the viewports (by initiating the Viewport Configuration CSC) and the 2-D overlays (by initiating the 2-D Overlay Compiler CSC). db\_mcc\_setup also loads the terrain database and the dynamic elements database (DED) into active area memory, and processes requests to download trajectory table data. Upon another state change request from the Simulation Host, db\_mcc\_setup calls simulation to start the simulation.

During the simulation, the process\_a\_msg function (called by simulation) processes all runtime messages. These messages may ask to move or rotate dynamic vehicles, change gun overlays, pass process round and round fired messages to Ballistics, or change other simulation parameters. process\_a\_msg also processes the hit and miss messages returned by Ballistics.

As the simulated vehicle moves about in active area memory, the database management functions read new load modules in from the terrain database as necessary. The database feedback functions prepare messages describing the terrain around the simulated vehicle and return them to the Simulation Host at specified intervals.

When the Simulation Host sends a message ending the simulation, control is passed back to db\_mcc\_setup. db\_mcc\_setup then initializes the configuration tree and returns control to upstart.

The CSUs in the Real-Time Processing CSC are identified in Figure 2-17 and are described in this section.

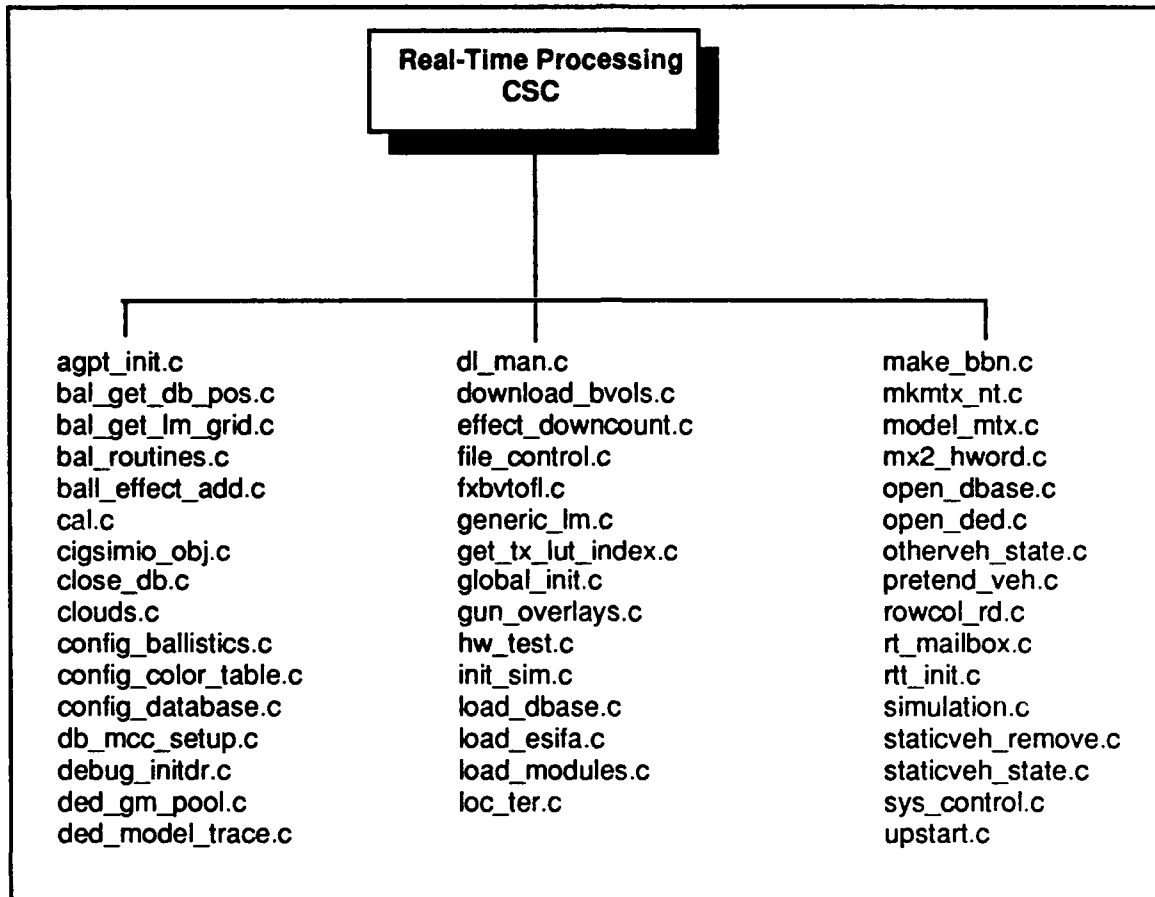


Figure 2-17. Real-Time Processing CSUs

### 2.13.1 agpt\_init.c

The agpt\_init function is not used in the standard GT100 system.

### 2.13.2 bal\_get\_db\_pos.c

The bal\_get\_db\_pos function finds the load module number and grid number of a given chord point. This function is used to determine the load module and grid of the simulated vehicle's current position.

The function call is **bal\_get\_db\_pos(pcrd, lm\_width, lm\_per\_side)**, where:

*pcrd* is a pointer to the chord data

*lm\_width* is the width of a load module

*lm\_per\_side* is the number of load modules in a row or column of AAM

bal\_get\_db\_pos uses the FIND\_LM macro (described in Appendix B) to determine the load module for the x and y coordinates specified in the chord data. It then calculates which grid the vehicle occupies within the load module. The load module and grid number are placed in the chord data structure.

Called By: local\_terrain

Routines Called: FIND\_LM

Parameters:	CHORD_DATA	*pcrd
	INT_4	lm_width
	INT_4	lm_per_side

Returns: none

### 2.13.3 bal\_get\_lm\_grid.c

The `bal_get_lm_grid` function finds the load modules and grids in the database that are intersected by a chord. This function is used to determine which four grids lie around the simulated vehicle.

The function call is `bal_get_lm_grid(pcrd, lm_per_side, lm_size, lm_base_addr, bal_search, dvl_search, lm_width)`, where:

*pcrd* is a pointer to the chord data  
*lm\_per\_side* is the number of load modules in a row or column of AAM  
*lm\_size* is the size in bytes of a load module  
*lm\_base\_addr* is the load module's base address  
*bal\_search* is the array in which to store load module offsets and grid words  
*dvl\_search* is the array in which to store dynamic module path data  
*lm\_width* is the width of a load module

The function returns TRUE if it is successful. It returns FALSE if an illegal chord (one longer than 125 meters) is detected.

Called By: local\_terrain

Routines Called: none

Parameters:	CHORD_DATA	pcrd[]
	INT_4	lm_per_side
	INT_4	lm_size
	INT_4	lm_base_addr
	SEARCH_LIST	bal_search[]
	INT_4	dvl_search[]
	INT_4	lm_width

Returns: 1 (TRUE)  
0 (FALSE)

#### 2.13.4 bal\_routines.c

The functions in the bal\_routines.c CSU handle the interface between the real-time software and the Ballistics Processing CSC. Many of these routines process messages passed to and from Ballistics. These functions are:

- sim\_bal\_init
- sim\_bal\_start
- sim\_bal\_frame\_rate
- sim\_bal\_req\_pt\_info
- sim\_bal\_agl\_wanted
- sim\_bal\_process\_msg
- sim\_bal\_process\_tracer
- sim\_bal\_traj\_chord
- sim\_bal\_round\_fired
- sim\_bal\_static\_add
- sim\_bal\_static\_rem
- sim\_bal\_reset
- sim\_bal\_tf\_veh\_update

##### 2.13.4.1 sim\_bal\_init

The sim\_bal\_init function initializes variables used in Ballistics messages. This function is called at the start of a simulation.

The function call is **sim\_bal\_init()**. sim\_bal\_init initializes the following:

- Variables used when processing rounds (estimated impact time and range).
- Parameters used in the Ballistics "new frame" message.
- The database index.
- Variables used when processing chords.

Called By: init\_simulation

Routines Called: none

Parameters: none

Returns: none

##### 2.13.4.2 sim\_bal\_start

The sim\_bal\_start function puts Ballistics into the run state and tells it where active area memory is. This function is called at the start of a simulation.

The function call is **sim\_bal\_start(pdbase)**, where *pdbase* is a pointer to the primary database control block. **sim\_bal\_start** does the following:

- Sets the Ballistics state to **BX\_RUN**.
- Pushes a **MSG\_B0\_STATE\_CONTROL** message onto the Ballistics message queue.
- Calculates the coordinates of the southwest corner of active area memory by subtracting half of the AAM width from the simulated vehicle's coordinates.
- Pushes a **MSG\_B0\_AAM\_SW\_CORNER** message onto the Ballistics message queue.

Called By:            **init\_simulation**

Routines Called:    **GLOB**  
                      **mx\_push**  
                      **printf**

Parameters:            **DB\_INFO**                            **\*pdbase**

Returns:                **none**

#### **2.13.4.3    sim\_bal\_frame\_rate**

The **sim\_bal\_frame\_rate** function tells Ballistics the current CIG frame rate by pushing a **MSG\_B0\_CIG\_FRAME\_RATE** message onto the Ballistics message queue. This function is called when a simulation is started.

The function call is **sim\_bal\_frame\_rate()**.

Called By:            **init\_simulation**

Routines Called:    **GLOB**  
                      **mx\_push**

Parameters:            **none**

Returns:                **none**

#### **2.13.4.4    sim\_bal\_req\_pt\_info**

The **sim\_bal\_req\_pt\_info** function processes the **MSG\_REQUEST\_POINT\_INFO** message. This message is sent by the Simulation Host to obtain the terrain characteristics at a specified point.

The function call is `sim_bal_req_pt_info(p_msg)`, where *p\_msg* is a pointer to the `MSG_REQUEST_POINT_INFO` message. The function does the following:

- Builds the chord data required by Ballistics. The chord's starting x and y coordinates are taken from the message from the Simulation Host. The function sets the starting and ending z coordinates to 20000 and -1, respectively. The type and id are set to `GETZ_CHORD`.
- Pushes a `MSG_B0_TRAJ_CHORD` message onto the Ballistics message queue.

Called By: `process_a_msg`

Routines Called: `GLOB`  
`mx_push`

Parameters: `MSG_REQUEST_POINT_INFO` `*p_msg`

Returns: `none`

#### 2.13.4.5 `sim_bal_agl_wanted`

The `sim_bal_agl_wanted` function sends a `MSG_B0_TRAJ_CHORD` message to Ballistics to (1) calculate the simulated vehicle's altitude above ground level (AGL), and (2) calculate the collision chord for the simulated vehicle, based on how far and in which direction it has moved since the last frame. This function also sends new frame information (frame count and the current state of all dynamic models) to Ballistics. This function is called at the end of every frame.

The function call is `sim_bal_agl_wanted(pdynl)`, where *pdynl* is a pointer to the structure containing the current state of all dynamic models. The function does the following:

- If AGL processing is enabled and the frame count is greater than 4:
  - Sets the trajectory chord id and type to `AGL_CHORD`.
  - Sets the chord's beginning and ending x and y coordinates to the simulated vehicle's current x and y coordinates.
  - Sets the chord's beginning z coordinate to the simulated vehicle's current z coordinate.
  - Sets the chord's ending z coordinate to -1.0.
  - Pushes a `MSG_B0_TRAJ_CHORD` message onto the Ballistics message queue.
  - Sets the trajectory chord id and type to `COL_CHORD`.
  - Sets the chord's beginning x, y, and z coordinates to the simulated vehicle's current coordinates.
  - Sets the chord's ending x, y, and z coordinates to 2 times the simulated vehicle's current coordinate minus its previous position. (This is the location the vehicle will occupy in the next frame if it moves in the same direction and the same distance it moved in the last frame.)
  - Pushes a `MSG_B0_TRAJ_CHORD` message onto the Ballistics message queue.

- Sets the simulated vehicle's previous position to its new current position.
- Passes the new frame count and the state of all dynamic models to Ballistics by pushing a MSG\_B0\_NEW\_FRAME message onto the Ballistics message queue.

Called By: msg\_end

Routines Called: GLOB  
mx\_push

Parameters: INT\_4 \*pdynl

Returns: none

#### 2.13.4.6 sim\_bal\_process\_msg

The sim\_bal\_process\_msg function is responsible for processing the messages that are returned to the real-time software from Ballistics. This function is called at the end of every frame, to generate the messages to be returned to the Simulation Host in the next exchange packet.

The function call is **sim\_bal\_process\_msg()**. The function does the following:

- Resets the count of terrain feedback points to zero.
- Previews and processes the top message in the Ballistics message queue.
- Deletes that message from the queue and previews the next.

The following table identifies the messages processed by sim\_bal\_process\_msg, and summarizes the processing performed for each one. The SEND\_TF\_INFO macro referenced in the table is described in Appendix B.

Message	Processing by <code>sim_bal_process_msg</code>
MSG_B1_HIT_RETURN AGL_CHORD COL_CHORD GETZ_CHORD  default	<i>Reports an intersection.</i> Sets the vehicle's altitude to its z coordinate. Sets the vehicle's altitude to -9999. Builds a MSG_RETURN_PT_INFO message with the intersection point data provided by Ballistics. Builds a MSG_HIT_RETURN message with the data provided by Ballistics.
MSG_B1_MISS AGL_CHORD COL_CHORD GETZ_CHORD default	<i>Reports a miss.</i> Sets the vehicle's altitude to minus its z coordinate. No action. No action. Builds a MSG_MISS message with the data provided by Ballistics.
MSG_B1_SHOT_REPORT	<i>Reports on rounds designated for specific targets.</i> Builds a MSG_SHOT_REPORT message.
MSG_B1_TF_HDR	<i>Terrain feedback header message.</i> Uses the SEND_TF_INFO macro to build a MSG_TF_HDR message.
MSG_B1_TF_PT	<i>Reports terrain feedback for a specified point.</i> Uses the SEND_TF_INFO macro to build one MSG_TF_PT message for each feedback point specified in the header message.

Called By:            `msg_end`

Routines Called:    `CLOB`  
                      `mx_peek`  
                      `mx_skip`  
                      `printf`  
                      `SEND_TF_INFO`

Parameters:            none

Returns:                none

#### 2.13.4.7    `sim_bal_process_tracer`

The `sim_bal_process_tracer` function processes tracer (round position) messages returned by Ballistics. This function is called at the end of every frame.

The function call is `sim_bal_process_tracer(db, pdbase)`, where:

*db* is the current double-buffer pointer

*pdbase* is a pointer to the primary database control block

`sim_bal_process_tracer` does the following:



- Previews the top message in the Ballistics message queue.
- If the message is a MSG\_B1\_ROUND\_POSITION message:
  - Verifies there is room for the tracer data.
  - Calls ball\_effect\_add to store the effects data in memory for incorporating in the scene with other effects.
  - If separate updating is enabled for AAM2, calls ball\_effect\_add to store the effects data in memory for that AAM.
- Deletes this message from the queue and previews the next one.

Called By: msg\_end

Routines Called: ball\_effect\_add  
GLOB  
mx\_peek  
mx\_skip  
printf

Parameters: INT\_4 db  
DB\_INFO \*pdbase

Returns: none

#### 2.13.4.8 sim\_bal\_traj\_chord

The sim\_bal\_traj\_chord function sends the MSG\_B0\_TRAJ\_CHORD message to Ballistics to compute the intersection of a specified trajectory chord. This function is called when the Simulation Host sends a MSG\_TRAJ\_CHORD message.

The function call is sim\_bal\_traj\_chord(db, pdbase), where:

*db* is the current double-buffer pointer

*pdbase* is a pointer to the primary database control block

sim\_bal\_traj\_chord does the following:

- Pushes a MSG\_B0\_TRAJ\_CHORD message onto the Ballistics message queue.
- If the message requests tracer effects and there is room for the effects data, calls ball\_effect\_add to store the effects data in memory for incorporating in the scene with other effects.
- If separate updating is enabled for AAM2, calls ball\_effect\_add to store the effects data in memory for that AAM.

Called By: process\_a\_msg

Routines Called: ball\_effect\_add  
GLOB  
mx\_push

Parameters:           INT\_4                               db  
                  DB\_INFO                               \*pdbase

Returns:               none

#### 2.13.4.9    sim\_bal\_round\_fired

The `sim_bal_round_fired` function pushes a `MSG_B0_ROUND_FIRED` message onto the Ballistics message queue.

The function call is `sim_bal_round_fired(msg_P)`, where *msg\_P* is a pointer to the `MSG_B0_ROUND_FIRED` message.

This function is not currently used. When it receives a `MSG_ROUND_FIRED` message from the Simulation Host, `process_a_msg` pushes the message onto the Ballistics queue itself.

Called By:            none

Routines Called:     GLOB  
                      mx\_push

Parameters:           MSG\_ROUND\_FIRED               \*msg\_P

Returns:              none

#### 2.13.4.10   sim\_bal\_static\_add

The `sim_bal_static_add` function notifies Ballistics when the Simulation Host adds a static vehicle to active area memory via a `MSG_STATICVEH_STATE` message.

The function call is `sim_bal_static_add(to, from, type, rcnt, veh_is_tank)`, where:

*to* is a pointer to the static vehicle's entry in the database  
*from* is a pointer to the `MSG_STATICVEH_STATE` message  
*type* is the type of vehicle (taken from the message)  
*rcnt* is a register counter used to store the vehicle's load module  
*veh\_is\_tank* indicates whether the vehicle is a tank (taken from the message)

`sim_bal_static_add` builds a `MSG_B0_ADD_STATIC_VEHICLE` message, then pushes it onto the Ballistics message queue.

Called By:            staticveh\_state

Routines Called:	GLOB mx_error mx_push printf	
Parameters:	INT_4 INT_4 INT_2 INT_2 BYTE	*to *from type rcnt veh_is_tank
Returns:	none	

#### 2.13.4.11 sim\_bal\_static\_rem

The `sim_bal_static_rem` function notifies Ballistics when the Simulation Host deletes a static vehicle from active area memory via a `MSG_STATICVEH_REM` message.

The function call is `sim_bal_static_rem(from, rcnt)`, where:

*from* is a pointer to the `MSG_STATICVEH_REM` message  
*rcnt* is a register counter used to store the vehicle's load module

`sim_bal_static_rem` builds a `MSG_B0_DELETE_STATIC_VEHICLE` message, then pushes it onto the Ballistics message queue.

Called By:	staticveh_remove	
Routines Called:	GLOB mx_error mx_push printf	
Parameters:	INT_4 INT_2	*from rcnt
Returns:	none	

#### 2.13.4.12 sim\_bal\_reset

The `sim_bal_reset` function tells Ballistics to reinitialize itself by pushing a `MSG_B0_STATE_CONTROL` message onto the Ballistics message queue with the new state set to `BX_RESET`. This function is called when the Simulation Host sends a `MSG_CIG_CTL - C_STOP` message to stop the simulation.

The function call is **sim\_bal\_reset()**.

Called By: msg\_cig\_ctl

Routines Called: GLOB  
mx\_push

Parameters: none

Returns: none

#### 2.13.4.13 sim\_bal\_tf\_veh\_update

The **sim\_bal\_tf\_veh\_update** function notifies Ballistics of the new position of the simulated vehicle, for the purposes of terrain feedback reporting. This function is called whenever the simulated vehicle's current position is updated.

This function is called only if the terrain feedback vehicle is the simulated vehicle. The Simulation Host may send MSG\_TF\_VEHICLE\_POS messages to update the positions of other vehicles for which Ballistics is collecting terrain feedback data. Such messages are pushed onto the Ballistics message queue by process\_a\_msg.

The function call is **sim\_bal\_tf\_veh\_update(time\_stamp, pos, matrix)**, where:

*time\_stamp* is the time stamped in the MSG\_TF\_VEHICLE\_POS message  
*pos* is the simulated vehicle's new position  
*matrix* is the simulated vehicle's new rotation matrix

**sim\_bal\_tf\_veh\_update** does the following:

- Determines whether Ballistics needs to process terrain feedback for this vehicle during this frame.
- Builds a MSG\_B0\_TF\_VEHICLE\_POS message with the data from the message.
- Pushes the message onto the Ballistics message queue.

The function always returns 0.

Called By: process\_vppos

Routines Called: GLOB  
mx\_push

Parameters: UNS\_2                      time\_stamp  
R4P3D                                \*pos  
RTS4x3\_MTX                        \*matrix

Returns: 0

### 2.13.5 ball\_effect\_add.c

The `ball_effect_add` function adds single-frame special effects to the current effects list. This function is called by the routines responsible for processing `MSG_ROUND_POSITION` and `MSG_TRAJ_CHORD` messages.

The function call is `ball_effect_add(db, type, x, y, z, pdbase, vme_offset)`, where:

*db* is the double buffer to be updated

*type* is the type of special effect

*x*, *y*, and *z* are the world coordinates at which the effect is to be shown

*pdbase* is a pointer to the primary database control block

*vme\_offset* is the VME address offset to active area memory

`ball_effect_add` does the following:

- Uses the `FIND_LM` macro (described in Appendix B) to find the load module on which to place the Ballistics tracer.
- Puts the data in the register used to store effects. This data includes the effect's coordinates, number of stamps, number of frames, and load module number.

Called By: `sim_bal_process_tracer`  
`sim_bal_traj_chord`

Routines Called: `FIND_LM`

Parameters:	<code>INT_4</code>	<code>db</code>
	<code>INT_2</code>	<code>type</code>
	<code>REAL_4</code>	<code>x</code>
	<code>REAL_4</code>	<code>y</code>
	<code>REAL_4</code>	<code>z</code>
	<code>DB_INFO</code>	<code>*pdbase</code>
	<code>INT_4</code>	<code>vme_offset</code>

Returns: none

### 2.13.6 cal.c

The `cal` (calibration menu) function exercises the video monitors by placing a known pattern on all channels. This function is called when the Gossip user selects the 1 ("calibration menu") option from the Gossip main menu. `cal` presents a menu that lets the user turn the calibration image or gunner pixel overlay on or off. The user is then able to verify the accuracy of the image and take appropriate measures.

The function call is `cal()`. `cal` displays the "Calibration menu -->" prompt and uses `unbf_getchar` to get the user's entry.

The following table identifies the options presented on the calibration menu, and summarizes the steps performed by `cal` to process each selection. The menu is displayed if the user enters an invalid character.

Calibration Menu Option	Processing by cal
0 calibration images off	Sets <code>pcal_flag</code> to FALSE in both AAMs.
1 offset calibration images on	Calls <code>make_cal_matrices</code> and <code>make_cal_patterns</code> to create and display the offset calibration overlay. If AAM2 is present, copies the overlay there.
2 color calibration pixels on	Calls <code>make_cal_matrices</code> and <code>gos_polys</code> to create and display the color calibration pixels. If AAM2 is present, copies the overlay there.
3 bbn logo on	Calls <code>make_cal_matrices</code> and <code>make_bbn_logo</code> to create and display the BBN logo overlay. If AAM2 is present, copies the overlay there.
5 calibration pixels on	Calls <code>backend_get_object_addr</code> to get the address of each backend. For a T backend, sets the pixel state to TRUE and calls <code>msg_ppm_pixel_state</code> .
6 calibration pixels off	Calls <code>backend_get_object_addr</code> to get the address of each backend. For a T backend, sets the pixel state to FALSE and calls <code>msg_ppm_pixel_state</code> .
x exit	Exits.

Called By: `gossip_tick`

Routines Called: `backend_get_object_addr`  
`bcopy`  
`GLOB`  
`gos_polys`  
`make_bbn_logo`  
`make_cal_matrices`  
`make_cal_patterns`  
`msg_ppm_pixel_state`  
`printf`  
`unbf_getchar`  
`VME_TO_VMX`

Parameters: `none`

Returns: `none`

### 2.13.7 cigsimio\_obj.c

The functions in the cigsimio\_obj.c CSU accommodate writing CIG-to-SIM and SIM-to-CIG messages to a binary disk file (cigsimio.000) during runtime. This option, called recording, can be enabled using the r ("record i/f messages menu") option on the Gossip main menu. The user can open or close the record file, start or stop recording, and set the number of frames to be recorded. Messages can be recorded during any CIG state, including simulation.

If recording is enabled, the cigsimio\_obj functions write *every* message each frame into a temporary buffer. Incoming messages are written to the buffer as they are received. Outgoing messages are written to the buffer when the packet is sent. At the same time, the contents of the buffer are written to a disk file (cigsimio.000).

The record file can be played back to repeat the recorded simulation exercise. This feature, called absolute script playback, is available through options on the Flea Script menu. Refer to section 2.7.19 for details.

In addition to (or instead of) writing messages to a disk file, messages can be sent to stdout each frame. This feature, called debug message display, is provided by the Host Interface debug routines. The Host Interface debug routines are dependent upon the cigsimio\_obj functions, in that they display messages from the temporary buffer created by the cigsimio\_obj functions. This interface works as follows:

- The recording and debug display features can be enabled through Gossip. The user can also select which message types are to be displayed.
- Each frame, the cigsimio\_write function copies the messages to a temporary buffer. If message recording is enabled, cigsimio\_write writes the buffer contents to disk also. If only debug display is enabled, the messages are written to the buffer only.
- At the end of each frame, the host\_if\_debug function waits for a mailbox message to be posted by cigsimio\_write. It then gets a pointer to the cigsimio buffer and calls host\_if\_debug\_tick to display the messages using the print\_msg\_\* functions. Only those messages that have been enabled for debug display are displayed; other messages in the buffer are ignored.
- The cigsimio buffer is initialized at the beginning of every frame.

The cigsimio\_obj functions use ifx\_write to write the messages to the binary disk file. The print\_msg\_\* functions that write the messages to stdout are tailored for each message type to provide formatted, readable displays.

For more information on debug message display, refer to the Host Interface Manager CSC. For more information on the specialized print\_msg\_\* functions, refer to the Message Processing CSC.

The functions in the cigsimio\_obj.c CSU are:

- gos\_cigsimio
- cigsimio\_msg\_in
- cigsimio\_msg\_out

- cigsimio\_write
- cigsimio\_get\_data
- cigsimio\_buffer\_init
- cigsimio\_frame\_end

### 2.13.7.1 gos\_cigsimio

The gos\_cigsimio function lets the Gossip user control the process of writing all messages to a disk file. This function is called if the user selects the r ("record i/f messages menu") option from the Gossip main menu.

The function call is gos\_cigsimio(). gos\_cigsimio displays a menu of options, and uses unbf\_getchar to get the keystroke entered by the user. The following table identifies these options and lists the major steps performed by gos\_cigsimio to process each selection.

Record I/F Messages Menu Option	Processing by gos_cigsimio
? print this menu	Displays the list of valid options.
b begin recording	Verifies that the parameters have been initialized; sets cigsimio_flag to TRUE.
c close record file	If cigsimio_open_flag is TRUE, sets cigsimio_flag and cigsimio_open_flag to FALSE; calls ifx_close to close the file. If cigsimio_open_flag is FALSE, displays an error.
e end recording	If cigsimio_flag is TRUE, sets cigsimio_close_flag to TRUE; if not, outputs an error.
f # frames written	Displays the number of frames written, maximum number of frames, and number of bytes written.
i initialize	Prompts the user for the file name; sets cigsimio_fn; initializes the frame and byte count parameters; sets cigsimio_init_flag to TRUE.
m change frames to record	Prompts the user for the number of frames to record; sets max_num_io_frames to the user's entry or to a default value if the entry exceeds the maximum allowed.
o open record file	Calls ifx_open to create and open the file; sets cigsimio_open_flag to TRUE.
q quit	Exits.
s write string to file	Prompts the user for the tag string; calls cigsimio_write to write the string to the file.
x exit	Exits.

Called By: gossip\_tick

Routines Called: cigsimio\_write  
ifx\_close  
ifx\_open



printf  
scanf  
strlen  
unbf\_getchar

Parameters: none

Returns: none

### 2.13.7.2 cigsimio\_msg\_in

The `cigsimio_msg_in` function determines whether or not an incoming message should be written to the `cigsimio` buffer. This function is called every time a message is read from an incoming exchange packet.

The function call is `cigsimio_msg_in(p_addr, byte_count, frame_flag)`, where:

*p\_addr* is a pointer to the incoming message  
*byte\_count* is the number of bytes in the message  
*frame\_flag* is <<TBD>>

`cigsimio_msg_in` first determines whether message recording or debug message display has been enabled through Gossip (i.e., if either `cigsimio_flag` or `cigsimio_log_request` is TRUE). If so, it calls `cigsimio_write` to copy the message to the buffer and, if recording is enabled, to write it to the disk file. The message is tagged TAG\_MSG\_TO\_CIG.

Called By: cig\_config  
db\_mcc\_setup  
file\_control  
get\_msg\_2d  
hw\_test  
process\_a\_msg  
upstart

Routines Called: cigsimio\_write

Parameters:	UNS_4	*p_addr
	INT_4	byte_count
	BOOLEAN	frame_flag

Returns: none

### 2.13.7.3 cigsimio\_msg\_out

The `cigsimio_msg_out` function determines whether or not an outgoing message packet should be written to the `cigsimio` buffer. During a simulation, this function is called at the

end of each frame. During other states, it is called whenever a MSG\_END message is received from the Simulation Host, indicating the end of a message packet.

The function call is `cigsimio_msg_out(p_addr, byte_count, frame_flag)`, where:

*p\_addr* is a pointer to the message packet  
*byte\_count* is the number of bytes in the packet  
*frame\_flag* is <<TBD>>

`cigsimio_msg_out` first determines whether message recording or debug message display has been enabled through Gossip (i.e., if either `cigsimio_flag` or `cigsimio_log_request` is TRUE). If so, it calls `cigsimio_write` to copy the messages to the buffer and, if recording is enabled, to write them to the disk file. The message packet is tagged TAG\_MSG\_TO\_CIG.

Called By:            `_set_up_for_next_frame`  
                       `cig_config`  
                       `db_mcc_setup`  
                       `file_control`  
                       `get_msg_2d`  
                       `hw_test`  
                       `upstart`

Routines Called:    `cigsimio_write`

Parameters:           `UNS_4`                                `*p_addr`  
                       `INT_4`                                `byte_count`  
                       `BOOLEAN`                           `frame_flag`

Returns:                none

#### 2.13.7.4    `cigsimio_write`

The `cigsimio_write` function writes incoming and outgoing messages to a temporary buffer and, if recording is enabled, to a file on disk. This function is called every time a message is received or a message packet is sent, but only if message recording and/or debug message display has been enabled through Gossip.

`cigsimio_write` also manages the record file. For example, it closes the file if it reaches its maximum size or the specified number of frames, and opens or closes the file per the user's request through Gossip.

The function call is `cigsimio_write(p_addr, byte_count, frame_flag, record_tag)`, where:

*p\_addr* is a pointer to the message or packet to be written  
*byte\_count* is the number of bytes to be written  
*frame\_flag* is <<TBD>>  
*record\_tag* identifies the type of message: CIG-to-SIM or SIM-to-CIG

cigsimio\_write does the following:

- If the Gossip user has requested that the record file be closed:
  - Sets cigsimio\_close\_flag and cigsimio\_open\_flag to FALSE.
  - Calls ifx\_close to close the record file.
  - Sets cigsimio\_flag to FALSE.
- If the record file has reached its maximum size:
  - Disables recording by setting cigsimio\_flag to FALSE.
  - Calls ifx\_close to close the record file.
  - Sets cigsimio\_open\_flag to FALSE.
- If the record file is not open:
  - Calls ifx\_open to open the file.
  - Sets cigsimio\_open\_flag to TRUE.
- Creates a new tag record (specifying the number of frames written, current time, and byte count) and copies it to the cigsimio buffer.
- Copies the message(s) to the cigsimio buffer.
- If recording is enabled and either frame\_flag is TRUE or the record tag is TAG\_MSG\_FROM\_CIG:
  - Writes the messages from the buffer to the record file on disk.
  - Increments the number of frames written.
  - If the maximum number of frames has been written, closes the record file and turns recording off.

Called By:           cigsimio\_msg\_in  
                      cigsimio\_msg\_out  
                      gos\_cigsimio

Routines Called:    bcopy  
                      ifx\_close  
                      ifx\_open  
                      ifx\_write  
                      read\_watch

Parameters:	UNS_4	*p_addr
	INT_4	byte_count
	BOOLEAN	frame_flag
	UNS_1	record_tag

Returns:            none

### 2.13.7.5    cigsimio\_get\_data

The cigsimio\_get\_data function returns a pointer to the cigsimio buffer. This function is called when the Host Interface debug display routines are ready to print the buffered messages to stdout.

The function call is **cigsimio\_get\_data(pkt\_len)**, where *pkt\_len* is a pointer for the size of the buffer.

`cigsimio_get_data` returns a pointer to the buffer and places the buffer size in the location specified by `pkt_len`.

Called By:	host_if_debug	
Routines Called:	none	
Parameters:	INT_4	*pkt_len
Returns:	*todisk_buffer	

#### 2.13.7.6 `cigsimio_buffer_init`

The `cigsimio_buffer_init` function initializes the `cigsimio` buffer and sets the `cigsimio_log_request` variable to TRUE (indicating that debug display is enabled). This function is called at the beginning of every frame, to clear the buffer for the new messages from the current frame.

The function call is `cigsimio_buffer_init()`.

Called By:	host_if_debug
Routines Called:	none
Parameters:	none
Returns:	none

#### 2.13.7.7 `cigsimio_frame_end`

The `cigsimio_frame_end` function sets the `cigsimio_log_request` variable to FALSE, then posts a message to the `cigsimio_complete` mailbox. This message tells `host_if_debug` that all messages from the previous frame have been written to the `cigsimio` buffer and are ready to be processed by `host_if_debug_tick`. This function is called at the end of every frame.

The function call is `cigsimio_frame_end()`.

Called By:	msg_end
Routines Called:	sc_post

Parameters: none

Returns: none

### 2.13.8 close\_db.c

The close\_db function closes and frees all database information, including names, headers, model address tables, and catalog tables. It also closes the terrain database and the DED file. This function is called when the Simulation Host sends a CIG Control-Stop message to end the simulation.

The function call is **close\_db()**. If the system contains multiple backends, close\_db frees and closes the database in each one.

Called By: msg\_cig\_ctl

Routines Called: free  
XCLOSE

Parameters: none

Returns: none

### 2.13.9 clouds.c

The functions in the clouds.c CSU activate and manage cloud models. These functions are:

- cloud\_init
- cloud\_update
- cloud\_update\_model
- cloud\_mgmt
- cloud\_placement
- cloud\_scud
- set\_lut
- rnd

#### 2.13.9.1 cloud\_init

The cloud\_init function sets up and initializes the cloud model and the cloud control structure. This function is called at viewport configuration time.

The function call is **cloud\_init(type)**, where *type* is the type of cloud model: **CLOUD\_TOP** or **CLOUD\_BOT**. cloud\_init does the following:

- Allocates and initializes the model call.

- Allocates and sets up the cloud control structure.

Called By:           cig\_config

Routines Called:    aam\_malloc  
                      GLOB  
                      INIT\_MTX

Parameters:           WORD                           type

Returns:             none

### 2.13.9.2    cloud\_update

The `cloud_update` function updates cloud parameters. This function is called when the Simulation Host sends a `MSG_CLOUD_STATE` message, which specifies the cloud top and bottom models and positions them at the desired altitudes.

The function call is `cloud_update(pmsg)`, where *pmsg* is a pointer to the `MSG_CLOUD_STATE` message. `cloud_update` does the following:

- If the cloud state in the message is FALSE (0):
  - Sets `clouds_wanted_G` to FALSE.
  - Turns off the cloud models.
  - If AAM2 exists, disables cloud processing there also.
  - Sets `cloud_state` to NULL.
- If the cloud state in the message is TRUE (1):
  - Validates the data in the message:
    - \* If the cloud bottom in the message is below the top, sets the bottom equal to the top.
    - \* If scud processing is enabled and the scud top and bottom are not outside the cloud top and bottom, sets the scud bottom to 5.0 meters below the cloud bottom, and sets the scud top to 5.0 meters above the cloud top.
  - Sets the new cloud (and scud) top and calls `cloud_update_model` to set the model's address.
  - Sets the new cloud (and scud) bottom and calls `cloud_update_model` to set the model's address.
  - Sets the scud top and bottom thickness parameters (the distance from the scud top to the cloud top and from the scud bottom to the cloud bottom).
  - Sets `new_cloud_data` to TRUE. This triggers `cloud_mgmt` to process the changes.

Called By:           process\_a\_msg

Routines Called:    AAM2\_ADDR  
                      cloud\_update\_model

GLOB

Parameters: MSG\_CLOUD\_STATE \*pmsg

Returns: none

### 2.13.9.3 cloud\_update\_model

The cloud\_update\_model function sets up a cloud model's address.

The function call is **cloud\_update\_model(type, model\_id)**, where:

*type* is **CLOUD\_TOP** or **CLOUD\_BOT**

*model\_id* is the model id from the Simulation Host message

cloud\_update\_model gets the model's address from the DED model address table, then puts the model's address into the cloud control structure. If the system contains two backends, cloud\_update\_model updates both databases.

Called By: cloud\_update

Routines Called: printf

Parameters: WORD type  
HWORD model\_id

Returns: none

### 2.13.9.4 cloud\_mgmt

The cloud\_mgmt function manages cloud placement and effects. This function is called at the end of every frame to process the changes requested during that frame.

The function call is **cloud\_mgmt()**. cloud\_mgmt does the following:

- If the cloud has changed altitude (i.e., new\_cloud\_data is TRUE):
  - Sets new\_cloud\_data to FALSE.
  - Updates the top and bottom models' information.
- If the simulated vehicle is now below the clouds:
  - Sets the cloud state to BELOW\_CLOUDS.
  - Turns on the cloud bottom model.
  - If AAM2 is present, turns on the cloud bottom model there also.
  - If the simulated vehicle is now in the bottom scud layer, calculates the scud ratio and calls cloud\_scud.
  - If the vehicle is not in the bottom scud layer, calls cloud\_scud with the scud ratio set to NO\_SCUD.

- Calls cloud\_placement to update the model's position.
- If the simulated vehicle is now above the clouds:
  - Sets the cloud state to ABOVE\_CLOUDS.
  - Turns on the cloud top model.
  - If AAM2 is present, turns on the cloud top model there also.
  - If the simulated vehicle is now in the top scud layer, calculates the scud ratio and calls cloud\_scud.
  - If the vehicle is not in the top scud layer, calls cloud\_scud with the scud ratio set to NO\_SCUD.
  - Calls cloud\_placement to update the model's position.
- If the simulated vehicle is now in the clouds:
  - Turns off the top or bottom model (whichever was active, based on the previous cloud state).
  - If AAM2 is present, turns the cloud model off there also.
  - Calls cloud\_scud with the scud ratio set to FULL\_SCUD.
  - Sets the cloud state to IN\_THE\_CLOUDS.

Called By: msg\_end

Routines Called: AAM2\_ADDR  
cloud\_placement  
cloud\_scud  
GLOB

Parameters: none

Returns: none

#### 2.13.9.5 cloud\_placement

The cloud\_placement function updates the cloud model's position. This function is called to move the cloud model to keep it in line with the simulated vehicle. The cloud is moved in a fixed increment (cloud\_modulo) set in the clouds.h include file.

The function call is **cloud\_placement(pcloud)**, where *pcloud* is a pointer to the cloud's control information.

cloud\_placement checks the simulated vehicle's new position, moves the cloud model in the required direction, and updates the cloud model's matrix accordingly.

Called By: cloud\_mgmt

Routines Called: GLOB

Parameters: CLOUD\_CONTROL \*pcloud



Returns: none

### 2.13.9.6 cloud\_scud

The `cloud_scud` function creates a scud effect that depends on the altitude of the simulated vehicle in comparison to the cloud and scud layer altitudes.

The function call is `cloud_scud(ratio)`, where *ratio* is one of the following:

- If the simulated vehicle is in a scud layer, the ratio is the distance between the vehicle and the edge of the scud layer, divided by the thickness of the scud layer.
- If the simulated vehicle is in the clouds, the ratio is `FULL_SCUD`.
- If the simulated vehicle is not in a scud layer or in the clouds, the ratio is `NO_SCUD`.

`cloud_scud` calls `set_lut` to select a lookup table entry suitable for the required scud effect. If the ratio is larger than the random number returned by `rnd`, it is treated as `FULL_SCUD`.

Called By: cloud\_mgmt

Routines Called: rnd  
set\_lut

Parameters: REAL\_4 ratio

Returns: none

### 2.13.9.7 set\_lut

The `set_lut` function selects an entry from the 3-D lookup table (`lut3d`) to display a scud effect. The entry selected is based on whether the simulated vehicle is in the clouds, in a scud layer, or completely above or below the clouds and scud layers.

The function call is `set_lut(n, idx)`, where:

*n* is the backend id (0 or 1)

*idx* is index of the desired lookup table

Called By: cloud\_scud

Routines Called: none

Parameters: INT\_4 n  
INT\_4 idx

Returns: none

### 2.13.9.8 rnd

The `rnd` function is a random number generator. This function is used to generate a number to compare to the scud ratio calculated when the simulated vehicle is inside a scud layer. If the ratio exceeds the random number generated, the `FULL_SCUD` lookup table entry is selected.

The function call is `rnd()`.

Called By: `cloud_scud`

Routines Called: none

Parameters: none

Returns: `seed * .0000000004656613`

### 2.13.10 config\_ballistics.c

The functions in the `config_ballistics.c` CSU are used to configure Ballistics and establish the communication path between Ballistics and the real-time software. These functions are:

- `config_ballistics`
- `bal_buffer_setup`

#### 2.13.10.1 config\_ballistics

The `config_ballistics` function reads the Ballistics configuration file (`ballist.cfg`). This file specifies whether to use a local Ballistics task or to interface with a Ballistics process running on another CPU. This function is called at initialization time.

The function call is `config_ballistics(tte, tid, base_priority)`, where:

*tte* is the Ballistics task's entry in the task table  
*tid* is the task id assigned to the Ballistics task  
*base\_priority* is the task's priority

`config_ballistics` does the following:

- Initializes the Ballistics board type to `NO BALLISTICS`.
- Checks to see if Ballistics processing has been disabled (by specifying the "b" argument on the startup command line). If so, sets `ball_is_present` to `FALSE` and exits.

- Sets ball\_is\_present to TRUE.
- Calls fopen to open the Ballistics configuration file.
  - If the file cannot be opened, or if the file specifies an invalid type, sets the Ballistics board type to MASTER147\_BALLISTICS.
  - If the file specifies Slave Ballistics, calls bus\_error to verify that the Slave board is present, then sets the Ballistics board type to SLAVE147\_BALLISTICS.
  - If the file specifies Master Ballistics, sets the Ballistics board type to MASTER147\_BALLISTICS.
- Calls fclose to close the Ballistics configuration file.
- If Master Ballistics is set, calls pcreate to create a task table entry for the Ballistics task.
- Pauses (using sc\_delay) to give the task time to start up.
- Calls bal\_buffer\_setup to establish an interface to the Ballistics task.
- Calls mx\_peek to get the MSG\_B1\_STATUS\_RETURN returned by Ballistics after it initializes, then discards the message.

If an error is detected, the function exits with a 1.

Called By:           main   (in rtt.c)

Routines Called:    bal\_buffer\_setup  
                       bus\_error  
                       exit  
                       fclose  
                       fgets  
                       fopen  
                       GLOB  
                       mx\_peek  
                       mx\_skip  
                       pcreate  
                       printf  
                       sc\_delay  
                       serror  
                       sscanf  
                       strcmp

Parameters:	<pre> struct {int      last;         int      in_use;         int      (*task_addr)();         char     *name;         int      priority_offset;         } INT_4 INT_4 </pre>	<pre> *tte tid base_priority </pre>
-------------	---	-------------------------------------

Returns:           none

### 2.13.10.2 bal\_buffer\_setup

The `bal_buffer_setup` function establishes the interface between the real-time software and Ballistics. The processes communicate via the Ballistics message queues, which are described in the Ballistics Processing CSC section.

The function call is `bal_buffer_setup()`. The function does the following:

- Pauses (using `sc_delay`) to wait for the Ballistics addresses to be published.
- Calls `mpv_nfind` to get the addresses and sizes of the three Ballistics message queues.
- If the addresses are not published yet, pauses and then calls `mpv_nfind` again. If it has 10 unsuccessful retries, `bal_buffer_setup` resets the Ballistics board type to `NO_BALLISTICS`, sets `ball_is_present` to `FALSE`, and quits.
- Gets the Ballistics queue parameters.
- If Ballistics printing is enabled, outputs the Ballistics message queue parameters to `stdout`.

The function returns `TRUE` if it is successful. It returns `FALSE` if `mpv_nfind` could not find the Ballistics message queue parameters.

Called By: `config_ballistics`

Routines Called: `GLOB`  
`mpv_nfind`  
`printf`  
`sc_delay`

Parameters: `none`

Returns: `1 (TRUE)`  
`0 (FALSE)`

### 2.13.11 config\_color\_table.c

The `config_color_table` function open a specified color table file. This function is called when the Simulation Host sends a `MSG_CIG_CTL - CIG_CONFIG` message. The name of the file can be specified with the `MSG_FILE_DESCR - DB_COLOR_CFG_0` (backend 0) or `DB_COLOR_CFG_1` (backend 1) message. The default file name is `color.cfg`.

The color table file contains the following:

- Color offset.
- Color description.
- 3-D lookup table (TX backends only).
- 2-D lookup table (TX backends only).
- ESIFA color (T backends only).

- Minimum acceptable fade value.
- Maximum acceptable fade value.
- DTP (thermal) switching value.

The function call is **config\_color\_table(pfn)**, where *pfn* is a pointer to the name of the color table file. **config\_color\_table** does the following:

- Opens the specified file.
- Gets and sets the values for all normal mode video registers.
- Gets and sets the values for all optional mode video registers.
- Gets and sets the values for video on/off.
- Gets and sets the values for dtp\_switching (normal, alternate, and modifier).
- Gets and sets the values for the fade bits (and\_sky, or\_sky, and\_laser, and or\_laser).
- Clears the color descriptions from memory.
- Reads the color table and sets the specified values in memory.
- Closes the file.

The function exits with a 1 if the specified file could not be opened or the table contains too many entries.

Called By: db\_mcc\_setup

Routines Called: exit  
fclose  
fgets  
fopen  
printf  
scanf

Parameters: BYTE \*pfn

Returns: none

### 2.13.12 config\_database.c

The functions in the config\_database.c CSU open the database configuration file. These functions are:

- config\_database
- errors
- func\_msg
- print\_pdbase

#### 2.13.12.1 config\_database

The config\_database function opens the database configuration (database.cfg) file to find and set the names of the terrain database and DED files, local terrain message parameters,

and terrain and field-of-view addresses. This function is called when the system is initialized.

A different database configuration file can be specified with the MSG\_FILE\_DESCR - DB\_SETUP\_0 (backend 0) or DB\_SETUP\_1 (backend 1) message. The database configuration file specifies the following:

- Database name.
- DED database name.
- Initial frame to return a local terrain message.
- Number of frames between subsequent local terrain messages.
- Terrain address and field-of-view table address for each AAM.

The function call is **config\_database()**. The function does the following:

- Initializes the primary database control blocks.
- Sets the default database and DED file names.
- Sets up the VME offsets for active area memory in each backend.
- Sets the default local terrain frequency.
- Calls errors to check for errors.
- Sets up pointers to the DATABASE\_MB mailboxes (one per rowcol\_rd task).
- If a subsys.cfg file exists:
  - Calls get\_database\_name to get the name of the database file for each backend.
  - Sets the db\_to\_use variable for each backend.
  - Calls get\_ded\_name to get the name of the DED file for each backend.
  - Sets the ded\_db\_to\_use variable for each backend.
  - Outputs the file names to stdout.
  - Calls errors to check for errors.
- If no subsys.cfg file is found:
  - Calls fopen to open the database.cfg file.
  - If no database.cfg file is found, outputs a message to stdout and returns a value indicating how many database management tasks need to be set up (one per backend).
  - If a database.cfg file is found, reads the configuration information (file names, local terrain message parameters, AAM terrain address, and AAM fov table address) for each database (each backend).
  - Calls errors to check for errors.
  - Closes the database.cfg file.

If no subsys.cfg or database.cfg file was found, the function returns 1 if one AAM is present, or 2 if two AAMs are present. If a configuration file was found, the function returns the number of AAMs as *i*. This triggers the creation of the correct number of rowcol\_rd tasks (one per backend).

The function exits with a 1 if an error is found.

Called By:           main   (in rtt.c)

Routines Called:    errors  
                      exit  
                      fclose

fgets  
 fopen  
 get\_database\_name  
 get\_ded\_name  
 printf  
 sprintf  
 sscanf  
 strcpy  
 strlen

Parameters: none

Returns: 1  
 2  
 i

### 2.13.12.2 errors

The errors function checks for the following errors for config\_database:

- Invalid terrain/field-of-view table addresses.
- Invalid local terrain frequency parameter.

The function call is **errors (pdbase)**, where *pdbase* is a pointer to the primary database control block.

The function returns TRUE if an error is found, or FALSE if the data is valid.

Called By: config\_database

Routines Called: bus\_error  
 func\_msg

Parameters: DB\_INFO \*pdbase

Returns: 1 (TRUE)  
 0 (FALSE)

### 2.13.12.3 func\_msg

The func\_msg function outputs the error "\*\*\*\* ERROR: <text>: DATABASE.CFG" to stdout. The variable text is supplied by the calling function.

The function call is **func\_msg(s)**, where *s* is the error text to appear in the output message.

Called By:	errors open_dbase
Routines Called:	printf
Parameters:	BYTE s[]
Returns:	none

#### 2.13.12.4 print\_pdbase

The `print_pdbase` function outputs the contents of a specified database configuration file to stdout. Specifically, it prints the following:

- Database identifier.
- Database file name.
- DED file name.
- Local terrain initial frame.
- Local terrain message frequency.
- Terrain address.
- Field-of-view table address.
- File descriptor.

The function call is `print_pdbase(pdb)`, where *pdb* is a pointer to the primary database control block.

This function is not currently used.

Called By:	none
Routines Called:	printf
Parameters:	DB_INFO *pdb
Returns:	none

#### 2.13.13 db\_mcc\_setup.c

The `db_mcc_setup` function processes messages from the Simulation Host to prepare the CIG system to run a simulation. It can also prepare the CIG to act as an MCC station, although this mode is not currently used. `db_mcc_setup` is called when the CIG Control message from the Simulation Host specifies `C_DB_SETUP` or `C_MCC_SETUP`.



The function call is **db\_mcc\_setup(state)**, where *state* is the state the system is to be set up in: C\_DB\_SETUP (simulation mode) or C\_MCC\_SETUP (MCC station mode).

db\_mcc\_setup does the following:

- Initializes trajectory table static variables.
- Sets the default ESIFA (textures) file name if none has been specified.
- Sets the default color configuration file name if none has been specified.
- Outputs the ESIFA and color file names to stdout.
- Calls poll\_shutdown to see if a system shutdown has been requested.
- Processes each message received from the Simulation Host:
  - Verifies that the packet length has not been exceeded; uses SYSERR to generate a MSG\_SYS\_ERROR if it has.
  - Calls cigsimio\_msg\_in to write the message to a buffer (if message recording or debug message display is enabled).
  - Processes the message based on its message type (see table below). Uses SYSERR to generate an error message if an invalid message is received.
- Returns control to upstart when it returns from cig\_config or a simulation, or when it detects a CIG-Control Stop message.

The following table summarizes the processing performed by db\_mcc\_setup in response to each valid message type it receives from the Simulation Host. The first column lists the messages in alphabetical order. The second column briefly describes the purpose of the message (in *italics*), then lists the major steps performed by db\_mcc\_setup to process the message.

Message from SIM Host	Processing by db_mcc_setup
MSG_ADD_TRAJ_TABLE	<i>Adds a trajectory table.</i> Calls mx_push to push MSG_B0_ADD_TRAJ_TABLE message onto Ballistics message queue.
MSG_AMMO_DEFINE	<i>Selects ammunition maps.</i> Loads specified ammo maps into memory.
MSG_CIG_CTL C_CIG_CONFIG  C_START_2D_SETUP C_MCC_SIMUL C_SIMULATION C_STOP C_NULL	<i>Causes a transition to another performance state.</i> Calls config_color_table; calls cig_config; calls load_dbase for each backend. Calls cig_2d_setup. Calls simulation with state set to C_MCC_SIMUL. Calls simulation with state set to C_SIMULATION. Returns to upstart. No action.
MSG_DEFINE_TX_MODE	<i>Defines viewport switching data for TX backends.</i> Calls mpvideo_define_mode.
MSG_DELETE_TRAJ_TABLE	<i>Deletes a trajectory table.</i> Calls mx_push to push MSG_B0_DELETE_TRAJ_TABLE message onto Ballistics message queue.
MSG_DR11_PKT_SIZE	<i>Specifies exchange packet parameters.</i> Outputs data from message; sets CIG and SIM exchange packet sizes; validates local terrain chunk size and sets if valid; validates local terrain message interval and sets if valid; sets CIG hardware type.

MSG_END	<i>Signals end of packet buffer.</i> Calls <code>cigsimio_msg_out</code> to write the outgoing message packet to the <code>cigsimio</code> buffer if recording or debug display is enabled; if <code>ppm_message_flag</code> is TRUE, calls <code>esifa_send_req</code> and sets flag to FALSE; calls the appropriate <code>exchange_data</code> routine (through <code>*exchange_data</code> ) to exchange packets.
MSG_FILE_DESCR	<i>Specifies database files to use for simulation.</i>
DB_SETUP	If no database has been specified, outputs a message; if specified, calls <code>open_dbase</code> .
DB_SETUP_0, DB_SETUP_1	Calls <code>set_database_name</code> for specified backend; gets and outputs file name.
DB_DED_SETUP_0, DB_DED_SETUP_1	Calls <code>set_ded_name</code> for specified backend; gets and outputs file name.
DB_COLOR_CFG_0, DB_COLOR_CFG_1	Calls <code>set_color_config_name</code> for both backends; gets and outputs file name.
DB_ESIFA_LOAD_0, DB_ESIFA_LOAD_1	Sets <code>esifa_load_wanted</code> to TRUE; outputs file name; calls <code>set_esifa_name</code> and <code>load_esifa</code> for specified backend.
DB_SUBSYS_0, DB_SUBSYS_1	Calls <code>init_subsys_parser</code> ; opens file; calls <code>set_token_file</code> ; calls <code>parse_subsys_file</code> ; closes file; outputs names of all database files for both backends; if <code>esifa</code> name is not set, sets it, sets <code>esifa_load_wanted</code> to TRUE, and calls <code>load_esifa</code> ; sets <code>db_to_use</code> and <code>ded_db_to_use</code> for both backends; calls <code>mpvideo_load</code> to load final lookup, 3d lookup, and <code>data_2d</code> tables for both backends.
DB_DATA_2D_0, DB_DATA_2D_1	Calls <code>set_data_2d_name</code> for specified backend; outputs file name; calls <code>mpvideo_load</code> for specified backend.
DB_FINAL_LUT_0, DB_FINAL_LUT_1	Calls <code>set_final_lut_name</code> for specified backend; outputs file name; calls <code>mpvideo_load</code> for specified backend.
DB_3D_LUT_0, DB_3D_LUT_1	Calls <code>set_3d_lut_name</code> for specified backend; outputs file name; calls <code>mpvideo_load</code> for specified backend.
MSG_PPM_DISPLAY_MODE	<i>Changes the PPM display modes.</i> Calls <code>msg_ppm_display_mode</code> ; sets <code>ppm_message_flag</code> to TRUE.
MSG_PPM_DISPLAY_OFFSET	<i>Changes the PPM display offsets.</i> Calls <code>msg_ppm_display_offset</code> ; sets <code>ppm_message_flag</code> to TRUE.
MSG_PPM_PIXEL_LOCATION	<i>Sets a new PPM pixel location.</i> Calls <code>msg_ppm_pixel_location</code> ; sets <code>ppm_message_flag</code> to TRUE.
MSG_PPM_PIXEL_STATE	<i>Turns PPM pixel processing on or off.</i> Calls <code>msg_ppm_pixel_state</code> ; sets <code>ppm_message_flag</code> to TRUE.
MSG_SIO_CLOSE	<i>Closes a serial I/O device.</i> Calls <code>sio_close</code> .
MSG_SIO_INIT	<i>Opens a serial I/O device.</i> Calls <code>sio_init</code> .

MSG_TF_INIT_HDR	<i>Controls how the CIG builds terrain feedback messages. Calls mx_push to push MSG_B0_TF_INIT_HDR message onto Ballistics message queue; if "own" vehicle, sets flags, vehicle id, counter, and frequency.</i>
MSG_TF_INIT_PT	<i>Specifies point for which terrain feedback is desired. Calls mx_push to push MSG_B0_TF_INIT_PT message onto Ballistics message queue.</i>
MSG_TF_STATE	<i>Enables/disables terrain feedback or changes frequency. Calls mx_push to push MSG_B0_TF_STATE message onto Ballistics message queue; if "own" vehicle, sets flags or changes frequency as indicated.</i>
MSG_TRAJ_ENTRY	<i>Adds an entry to a trajectory table. Calls mx_push to push MSG_B0_TRAJ_ENTRY message onto Ballistics message queue.</i>
MSG_TRAJ_ENTRY_XFER	<i>Downloads an entry to a Ballistics trajectory table. Verifies table has room; sets trajectory table entry data; calls mx_push to push MSG_B0_ADD_TRAJ_ENTRY message onto Ballistics message queue.</i>
MSG_TRAJ_TABLE_XFER	<i>Sets up a Ballistics trajectory table to be downloaded. Verifies no other table is in load process; sets data for trajectory table; calls mx_push to push MSG_B0_TRAJ_TABLE message onto Ballistics message queue.</i>

Called By:           upstart

Routines Called:    \*exchange\_data  
                       cig\_2d\_setup  
                       cig\_config  
                       cigsimio\_msg\_in  
                       cigsimio\_msg\_out  
                       config\_color\_table  
                       esifa\_send\_req  
                       fclose  
                       fopen  
                       get\_3d\_lut\_name  
                       get\_color\_config\_name  
                       get\_data\_2d\_name  
                       get\_database\_name  
                       get\_ded\_name  
                       get\_esifa\_name  
                       get\_final\_lut\_name  
                       GLOB  
                       init\_subsys\_parser  
                       load\_dbase  
                       load\_esifa  
                       mpvideo\_define\_mode  
                       mpvideo\_load  
                       msg\_ppm\_display\_mode  
                       msg\_ppm\_display\_offset  
                       msg\_ppm\_pixel\_location

msg\_ppm\_pixel\_state  
mx\_push  
open\_dbase  
parse\_subsys\_file  
poll\_shutdown  
printf  
set\_3d\_lut\_name  
set\_color\_config\_name  
set\_data\_2d\_name  
set\_database\_name  
set\_ded\_name  
set\_esifa\_name  
set\_final\_lut\_name  
set\_token\_file  
simulation  
sio\_close  
sio\_init  
start\_watch  
strcmp  
strcpy  
strlen  
strncmp  
SYSERR

Parameters: INT\_2 state

**Returns:** `none`

### 2.13.14 debug\_initdr.c

The `debug_initdr` function calls the `host_if_debug_tick` function to initiate printing/display of all enabled message types in the incoming packet buffer, for debugging or testing purposes.

The function call is **debug\_initdr()**.

**This function is not currently used.**

Called By: none

**Routines Called:** host\_if\_debug\_tick

Parameters: none

Returns: none

### 2.13.15 ded\_gm\_pool.c

The functions in the `ded_gm_pool.c` CSU are used to create and manage DED memory pools. The standard DED pool, pool 0, is used for 3500-meter databases. In 7000-meter databases with large DED files, a second pool (pool 1) can be allocated and used also. Using standard load module AAM to store extra DED models is called DED wrapping.

The functions in this CSU are:

- `pool_init`
- `pool_get_off_24`
- `addr_in_pool`

The `DED_POOL` structure, used to manage DED memory, specifies the pool's start address, next and last available bus addresses, DED size, total byte count, and the offset to add to model addresses.

#### 2.13.15.1 pool\_init

The `pool_init` function sets up DED pool 0. It also sets up pool 1 if it determines that the additional memory is available and required.

The function call is `pool_init (pdbase, avail_gm, file_cut, vme_offset)`, where:

*pdbase* is a pointer to the primary database control block  
*avail\_gm* is the amount of hardware memory available in standard generic memory  
*file\_cut* is the size of pool 0 or the address of the first model in pool 1  
*vme\_offset* is the VME address offset to active area memory

`pool_init` does the following:

- Set up DED pool 0's entries using the passed parameters.
- Initializes DED pool 1's entries to 0.
- Determines whether pool 0 is large enough for the amount of generic memory required.
- If pool 0 is not large enough:
  - If the database has a 7000-meter viewing range, verifies that pool 1 is large enough, then sets up pool 1's entries.
  - If the database does not have a 7000-meter viewing range, outputs an error.

The function returns `TRUE` if it successfully initializes pool 1 for additional generic memory. It returns `FALSE` if pool 0 is adequate, or if additional memory is required but is not available.

Called By: `open_ded`

Routines Called: `printf`

Parameters:	DB_INFO INT_4 INT_4 WORD	*pdbname avail_gm file_cut vme_offset
-------------	-----------------------------------	--

Returns:	1 (TRUE) 0 (FALSE)
----------	-----------------------

### 2.13.15.2 pool\_get\_off\_24

The `pool_get_off_24` function returns the appropriate 24-bit offset for a given model address.

The function call is `pool_get_off_24(file_addr)`, where *file\_addr* is the model address.

The function determines which pool the address is in by comparing it to the *file\_cut* (the first model address in pool 1). It then returns the offset to that pool.

Called By:	ded_model_trace
------------	-----------------

Routines Called:	none
------------------	------

Parameters:	INT_4	file_addr
-------------	-------	-----------

Returns:	pools[0].adr_offset & MASK_24 pools[1].adr_offset & MASK_24
----------	--

### 2.13.15.3 addr\_in\_pool

The `addr_in_pool` function returns TRUE if a specified address is valid (i.e., is within either DED pool).

The function call is `addr_in_pool(cmnd_addr, msg)`, where:

*cmnd\_addr* is the command address to be checked  
*msg* is TRUE if an error message is to be output if the address is valid, or FALSE if no output message is desired

If the *cmnd\_addr* is in either DED pool, the function returns TRUE.

If the *cmnd\_addr* is not in either pool, the function returns FALSE. If *msg* is TRUE, the function also outputs an error message showing the pool boundaries.

Called By:	ded_model_trace open_ded
------------	-----------------------------

Routines Called:	printf	
Parameters:	INT_4 BOOLEAN	cmd_addr msg
Returns:	1 (TRUE) 0 (FALSE)	

### 2.13.16 ded\_model\_trace.c

The `ded_model_trace` function traces the Data Traversal Processor (DTP) commands for each dynamic model and special effect, and adjusts addresses based on the commands.

The function call is `ded_model_trace(ded_size, ded_start_addr, model_start_addr, gm_end_addr, shift_addr, last_shift_addr)`, where:

*ded\_size* is the amount of memory available for all dynamic models (in both DED pools, if both are allocated)  
*ded\_start\_addr* is the starting address of the current DED pool  
*model\_start\_addr* is the starting address of the model to be traced  
*gm\_end\_addr* is the highest address in the current DED pool  
*shift\_addr* is the address to shift boundary data to  
*last\_shift\_addr* is the last possible address for command shifting

`ded_model_trace` does the following:

- Checks the model's start address to make sure it is in one of the DED pools.
- Steps through each DTP command for each model and each special effect.
- Resets the *shift\_addr*.

The function returns the final value of *shift\_addr* if it is successful. It returns -1 if the model's address is not in either DED pool.

The macros used by `ded_model_trace` (DED\_BOUNDARY, PUSH\_STACK, and POP\_STACK) are described in Appendix B.

Called By:	open_ded	
Routines Called:	addr_in_pool DED_BOUNDARY pool_get_off_24 POP_STACK printf PUSH_STACK	
Parameters:	INT_4 INT_4	ded_size ded_start_addr

INT\_4  
INT\_4  
INT\_4  
INT\_4

model\_start\_adr  
gm\_end\_adr  
shift\_adr  
last\_shift\_adr

Returns: shift\_adr  
-1

### 2.13.17 dl\_man.c (dl\_setup)

The dl\_setup function initializes various active area memory state tables.

The function call is **dl\_setup()**. dl\_setup does the following:

- Initializes tanks and other vehicles in the dynamic state table.
- Initializes static tanks and other static vehicles in the static state table.
- Initializes the multiple-frame effects linked list. (This structure is used when showing effects over multiple frames.)

Called By: active\_area\_init

Routines Called: INIT\_MTX

Parameters: none

Returns: none

### 2.13.18 download\_bvols.c

The download\_bvols function downloads model directories, model entries, and bounding volumes to Ballistics.

The function call is **download\_bvols(header\_P, fd, dev\_P, model\_type)**, where:

*header\_P* is a pointer to the database header data  
*fd* is an identifier for the file containing the information to be downloaded  
*dev\_P* is a pointer to the Ballistics message queue  
*model\_type* is BX\_DED\_MODEL\_DIRECTORY

download\_bvols does the following:

- Allocates memory to work in.
- For the model directory:
  - Builds a structure with the model directory data.
  - Passes the data to Ballistics by calling mx\_push to push a MSG\_B0\_MODEL\_DIRECTORY message onto the Ballistics message queue.



- For each model entry:
  - Reads the entry from the specified file.
  - Builds a structure with the model's data.
  - Passes the data to Ballistics by calling `mx_push` to push a `MSG_B0_MODEL_ENTRY` message onto the Ballistics message queue.
- Frees the work area memory.
- Reads and validates the bounding volume count from the database header.
- Allocates memory to work in.
- For each bounding volume entry:
  - Reads the entry from the specified file.
  - Converts the bounding volume from fixed point to floating point.
  - Builds a structure with the bvol's data.
  - Passes the data to Ballistics by calling `mx_push` to push a `MSG_B0_BVOL_ENTRY` message onto the Ballistics message queue.
- Frees the work area memory.

The function returns 0 if successful. It returns -1 if the number of bounding volumes is less than 0.

Called By: `open_ded`

Routines Called: `BCOPY`  
`free`  
`fxbvtofl_020`  
`malloc`  
`mx_error`  
`mx_push`  
`printf`  
`XLSEEK`  
`XREAD`

Parameters:	<code>DB_HDR_DBASE_DATA</code>	<code>*header_P</code>
	<code>int</code>	<code>fd</code>
	<code>MX_DEVICE</code>	<code>*dev_P</code>
	<code>BYTE</code>	<code>model_type</code>

Returns: `0`  
`-1`

### 2.13.19 `effect_downcount.c`

The `effect_downcount` function counts down the frames for each stamp in a multiple-frame special effect. Once the complete effect has been displayed, it is removed from the active effects table. This function is called at the end of every frame.

The function call is `effect_downcount(pdbase, vme_offset)`, where:

*pdbase* is the pointer to the database control block  
*vme\_offset* is the VME address offset to active area memory

The active effects table is maintained in the first backend's active area memory only.

Called By:	_downcount_effects		
Routines Called:	none		
Parameters:	DB_INFO	*pdbase	
	INT_4	vme_offset	
Returns:	none		

### 2.13.20 file\_control.c

The file\_control function processes messages from the Simulation Host to handle file transfers to and from the Simulation Host, delete files, and produce a CIG disk directory list for the Simulation Host. file\_control is called whenever the state requested by the Simulation Host is C\_FILE\_XFER.

The function call is **file\_control(state)**, where *state* is the current state of the CIG system (C\_FILE\_XFER).

file\_control does the following:

- Calls poll\_shutdown to see if a system shutdown has been requested.
- For each message received from the Simulation Host:
  - Verifies that the packet length has not been exceeded. Calls SYSERR to generate an error message if it has.
  - Calls cigsimio\_msg\_in to write the message to a buffer (if message recording or debug display is enabled).
  - Processes the message according to its type (see table below).

The following table summarizes the processing performed by file\_control in response to each valid message type it receives from the Simulation Host. The first column lists the messages in alphabetical order. The second column briefly describes the purpose of the message (in italicized letters), then lists the major steps performed by file\_control to process the message.

Message from SIM Host	Processing by file_control
MSG_CIG_CTL C_NULL C_STOP	<i>Causes a transition to another performance state.</i> No action. Returns control to upstart.
MSG_DR11_PKT_SIZE	<i>Specifies exchange packet parameters.</i> Sets CIG and SIM exchange packet sizes; validates local terrain chunk size and interval, and sets if valid; sets CIG hardware type.
MSG_END	<i>Signals end of packet buffer.</i> Calls cigsimio_msg_out to write outgoing message packet to buffer if recording is enabled; calls appropriate exchange_data routine (through *exchange_data) to exchange packets.
MSG_FILE_DESCR DB_CIG2SIM  DB_SIM2CIG  DB_DELETION DB_DIRECTORY  DB_REN_FROM DB_REN_TO	<i>Transfers and manages files.</i> Opens and reads file header; generates MSG_FILE_STATUS acknowledgement message; uses *exchange_data to exchange packets; transfers and checksums each block; generates MSG_FILE_XFER acknowledgement messages; uses *exchange_data to exchange packets; closes file. Calls create_sz to create file with specified name and size; generates MSG_FILE_STATUS acknowledgement message. Calls system to delete file from CIG disk; generates MSG_FILE_STATUS acknowledgement message. Passes CIG directory contents to SIM using MSG_FILE_DESCR messages; generates MSG_FILE_STATUS acknowledgement message. Finds file with specified name; generates MSG_FILE_STATUS acknowledgement message. Calls system to rename file to specified name; generates MSG_FILE_STATUS acknowledgement message.
MSG_FILE_STATUS	<i>Provides response for file transfer.</i> No action if file block was received; resends block or aborts if SIM reports error.
MSG_FILE_XFER	<i>Contains the data to upload or download.</i> Reads or writes data; generates MSG_FILE_STATUS return message.

Called By: upstart

Routines Called: \*exchange\_data  
cigsimio\_msg\_in  
cigsimio\_msg\_out  
close  
create\_sz  
GLOB  
lseek  
open  
poll\_shutdown

printf  
read  
rsec  
start\_watch  
strcpy  
strlen  
SYSERR  
system  
write

Parameters: INT\_2 state

Returns: none

### 2.13.21 fxbvtofl.c

The functions in the fxbvtofl.c CSU convert fixed point bounding volumes to floating point. These functions are:

- fxbvtofl\_dart
- fxbvtofl
- fxbvtofl\_020

#### 2.13.21.1 fxbvtofl\_dart

The fxbvtofl\_dart function converts a fixed point bounding volume's vertex list to floating point.

The function call is **fxbvtofl\_dart(tobv, frombv)**, where:

*tobv* is the floating point bvol entry  
*frombv* is the fixed point bvol entry

This function is not currently used.

Called By: none

Routines Called: FXTO881

Parameters: BVOL\_ENTRY \*tobv  
FIX\_BVOL\_ENTRY \*frombv

Returns: none

### 2.13.21.2 fxbvtofl

The fxbvtofl function converts a fixed point bounding volume's vertex list to floating point.

The function call is **fxbvtofl (tobv, frombv)**, where:

*tobv* is the floating point bvol entry  
*frombv* is the fixed point bvol entry

This function is not currently used.

Called By:	none	
Routines Called:	FXT0881	
Parameters:	BVOL_ENTRY FIX_BVOL_ENTRY	*tobv *frombv
Returns:	none	

### 2.13.21.3 fxbvtofl\_020

The fxbvtofl\_020 function converts a fixed point bounding volume's vertex list to floating point.

The function call is **fxbvtofl\_020(tobv, frombv)**, where:

*tobv* is the floating point bvol entry  
*frombv* is the fixed point bvol entry

Called By:	download_bvols	
Routines Called:	FXT0881	
Parameters:	BVOL_ENTRY FIX_BVOL_ENTRY	*tobv *frombv
Returns:	none	

### 2.13.22 generic\_lm.c

The functions in the generic\_lm.c CSU are used to initialize and generate a generic load module containing one ocean polygon. This allows a system to go beyond the defined database boundaries while still retaining some orientation reference.

These functions are:

- init\_generic\_lm
- generic\_lm

#### 2.13.22.1 init\_generic\_lm

The init\_generic\_lm function initializes a generic load module. It generates the load module header, the required DTP commands, and the grid components.

The function call is **init\_generic\_lm(db\_index, view\_range)**, where:

*db\_index* identifies the backend (0 or 1)

*view\_range* is the viewing distance in meters (3500 or 7000).

The function returns FALSE if the db\_index is not 0 or 1.

Called By: load\_modules

Routines Called: none

Parameters:	INT_4	db_index
	INT_4	view_range

Returns: 0 (FALSE)

#### 2.13.22.2 generic\_lm

The generic\_lm function generates the generic load module. It copies the load module to memory, then updates the load module header, DTP, and grid components.

The function call is **generic\_lm(db\_index, swx, swy, centoff, memptr)**, where:

*db\_index* identifies the backend (0 or 1)

*swx* is the x coordinate of the load module's southwest corner

*swy* is the y coordinate of the load module's southwest corner

*centoff* is the offset to the center of the load module

*memptr* is a pointer to the AAM location for the new load module

Called By:	getside	
Routines Called:	none	
Parameters:	INT_4 INT_4 INT_4 REAL_4 GENLM	db_index swx swy centoff *memptr
Returns:	none	

### 2.13.23 get\_tx\_lut\_index.c

The `get_tx_lut_index` function returns the desired final output color table index.

The function call is `get_tx_lut_index(pview_mode)`, where *pview\_mode* is a pointer to the system view mode. The function does the following:

- Isolates the sky color (sky).
- Isolates the time of day (day).
- Isolates the thermal flag (thm).
- If thm is 1 and the `thermal_mode_wanted` variable is TRUE, returns 10 (the default thermal). If not, returns (sky \* 3 + day).

This function is not currently used.

Called By:	none	
Routines Called:	GLOB	
Parameters:	INT_4	*pview_mode
Returns:	sky * 3 + day 10	

### 2.13.24 global\_init.c

The `global_init.c` CSU contains functions that initialize various global variables. These functions are:

- `host_if_buffer_init`
- `host_init_packet_sizes`

#### 2.13.24.1 host\_if\_buffer\_init

The `host_if_buffer_init` function initializes the output buffer header (packet version number and level) and, optionally, the input and output message buffers (INBUF and OUTBUF).

The function call is `host_if_buffer_init (init_INBUF_and_OUTBUF)`, where `init_INBUF_and_OUTBUF` is TRUE if the message buffers are to be initialized.

Called By:	init_mpv_interface initialize	
Routines Called:	none	
Parameters:	BOOLEAN	init_INBUF_and_OUTBUF
Returns:	none	

#### 2.13.24.2 host\_init\_packet\_sizes

The `host_init_packet_sizes` function initializes the following global variables to their default values, if they have not yet been set:

- DR11 incoming and outgoing exchange packet sizes.
- SCSI incoming and outgoing exchange packet sizes.
- Local terrain chunk size.
- Interval between local terrain messages.

The function call is `host_init_packet_sizes()`.

Called By:	initialize
Routines Called:	none
Parameters:	none
Returns:	none

#### 2.13.25 gun\_overlays.c

The functions in the `gun_overlays.c` CSU are used to build M1 and M2 overlays. These overlays are hard-coded displays of three-dimensional polygons that are displayed on the viewport, over the terrain display. The overlay shows objects that would normally obscure



the view of the terrain, to better emulate the real-world view out the vehicle's window. These overlays are vehicle-specific.

gun\_overlays.c contains the following functions:

- m1\_gun\_overlay
- m2\_gun\_overlay
- make\_m1\_overlays
- make\_m2\_overlays

These functions apply to T backends only. Overlays for TX backends are generated by the 2-D Overlay Compiler CSC using Simulation Host messages.

### 2.13.25.1 m1\_gun\_overlay

The m1\_gun\_overlay function creates gun and gunner overlays for M1 vehicles. This function is called when the message from the Simulation Host is MSG\_GUN\_OVERLAY and the message type is M1\_OVERLAYS.

Gun overlays show the components of the gun (on the simulation vehicle) that would be visible when looking out from the vehicle's window. Gunner overlays show cross-hairs and digits. The MSG\_GUN\_OVERLAY message specifies the digits to be displayed.

The function call is m1\_gun\_overlay(pmsg, db), where:

*pmsg* is a pointer to the MSG\_GUN\_OVERLAY message  
*db* is the double-buffer memory current base pointer

m1\_gun\_overlay does the following:

- Checks to see if an overlay has been defined.
- Updates the gun barrel viewport-to-turret transformation matrix.
- Updates the gunners overlay polygons.

Called By: process\_a\_msg

Routines Called: syserr

Parameters: MSG\_GUN\_OVERLAY \*pmsg  
INT\_4 db

Returns: none

### 2.13.25.2 m2\_gun\_overlay

The m2\_gun\_overlay function creates gun overlays for M2 vehicles. This function is called when the message from the Simulation Host is MSG\_GUN\_OVERLAY and the message type is M2\_OVERLAYS.

Gun overlays show the components of the gun (on the simulation vehicle) that would be visible when looking out from the vehicle's window. Gunner overlays show cross-hairs and digits. The MSG\_GUN\_OVERLAY message specifies the digits to be displayed.

The function call is **m2\_gun\_overlay(pmsg, db)**, where:

*pmsg* is a pointer to the MSG\_GUN\_OVERLAY message  
*db* is the double-buffer memory current base pointer

**m2\_gun\_overlay** does the following:

- Checks to see if an overlay has been defined.
- Updates the gun barrel viewport-to-turret transformation matrix.
- Updates the gunners overlay polygons.

Called By: process\_a\_msg

Routines Called: syserr

Parameters: MSG\_GUN\_OVERLAY      \*pmsg  
 INT\_4      db

Returns: none

### 2.13.25.3 make\_m1\_overlays

The **make\_m1\_overlays** function sets up M1 overlay data at viewport configuration time. This function is called if the Simulation Host sends a MSG\_OVERLAY\_SETUP message with the type set to 1 (M1\_OVERLAYS).

*Note: The MSG\_OVERLAY\_SETUP message can specify gunners\_viewport (the viewport that is to have the gunner's overlay) and barrel\_viewports (the viewports the gun barrel is to be viewable in). These values are not currently used. The gunner's overlay is placed on any viewport belonging to a configuration node that has bit 0 of its branch mask set. The gun barrel overlay is placed on any viewport belonging to a configuration node that has bit 1 of its branch mask set.*

The function call is **make\_m1\_overlays(po, ppg)**, where:

*po* is a pointer to the overlay parameters  
*ppg* is a pointer to the M1\_GUN\_OVERLAY message

**make\_m1\_overlays** does the following:

- Allocates memory for the overlay.
- Outputs the matrix, vertex list, digits, ASCII data, etc.

- Loads the overlay digit vertices.
- Sets up the gun tube overlay.
- Sets up the matrix command.
- Generates the gun barrel polygon component.
- Sets the overlay\_defined flag to TRUE.

Called By: overlay\_setup

Routines Called: aam\_malloc  
id\_4x3mtx  
make\_p\_nt  
swap\_axis

Parameters: OVERLAY\_PARAMS \*po  
M1\_GUN\_OVERLAY \*\*ppg

Returns: none

#### 2.13.25.4 make\_m2\_overlays

The make\_m2\_overlays routine sets up M2 overlay data at viewport configuration time. This function is called if the Simulation Host sends a MSG\_OVERLAY\_SETUP message with the message type set to 2 (M2\_OVERLAYS).

*Note: The MSG\_OVERLAY\_SETUP message can specify gunners\_viewport (the viewport that is to have the gunner's overlay) and barrel\_viewports (the viewports the gun barrel is to be viewable in). These values are not currently used. The gunner's overlay is placed on any viewport belonging to a configuration node that has bit 0 of its branch mask set. The gun barrel overlay is placed on any viewport belonging to a configuration node that has bit 1 of its branch mask set.*

The function call is make\_m2\_overlays(po, ppg), where:

po is a pointer to the overlay parameters  
ppg is a pointer to the M2\_GUN\_OVERLAY message

make\_m2\_overlays does the following:

- Allocates memory for the overlay.
- Builds the gunner display buffer.
- Loads the matrix.
- Outputs the gunner polygons.
- Loads the overlay digit vertices.
- Sets up the gun tube overlay.
- Sets up the matrix command.
- Generates the gun barrel's top, right, and left polygon components.
- Sets the overlay\_defined flag to TRUE.

Called By: overlay\_setup

Routines Called: aam\_malloc  
id\_4x3mtx  
make\_p\_nt  
swap\_axis

Parameters: OVERLAY\_PARAMS \*po  
M2\_GUN\_OVERLAY \*\*ppg

Returns: none

### 2.13.26 hw\_test.c

The hw\_test function processes messages from the Simulation Host to perform hardware tests. hw\_test is called whenever the state requested by the Simulation Host is C\_TEST\_MODE.

*Note: Hardware tests are not currently implemented.*

The function call is hw\_test(state), where state is the current state of the CIG system (C\_TEST\_MODE).

hw\_test does the following:

- Calls poll\_shutdown to see if a system shutdown has been requested.
- For each valid message type it receives:
  - Verifies that the packet length has not been exceeded; calls SYSERR to generate an error message if it has.
  - Calls cigsimio\_msg\_in to write the message to a buffer (if message recording or debug display is enabled).
  - Processes the message according to its type (see table below).

The following table summarizes the processing performed by hw\_test in response to each valid message type it receives from the Simulation Host. The first column lists the messages in alphabetical order. The second column briefly describes the purpose of the message (in *italics*), then lists the major steps performed by hw\_test to process the message.

Message from SIM Host	Processing by hw_test
MSG_CIG_CTL C_NULL C_STOP	<i>Causes a transition to another performance state.</i> No action. Returns to upstart.
MSG_DR11_PKT_SIZE	<i>Specifies exchange packet parameters.</i> Sets CIG and SIM exchange packet sizes; validates local terrain chunk size and message interval, and sets if valid; sets CIG hardware type.
MSG_END	<i>Signals end of packet buffer.</i> Calls cigsimio_msg_out to write outgoing message packet to buffer if recording is enabled; calls start_watch; calls appropriate exchange_data routine (through *exchange_data) to exchange packets.
MSG_TEST_NAME ECHO_PKT	<i>Specifies test to be run.</i> No action; test is not yet implemented.

Called By: upstart

Routines Called: \*exchange\_data  
cigsimio\_msg\_in  
cigsimio\_msg\_out  
GLOB  
poll\_shutdown  
start\_watch  
SYSERR

Parameters: INT\_2 state

Returns: none

### 2.13.27 init\_sim.c (init\_simulation)

The init\_simulation function initializes a new simulation. This function is called whenever the Simulation Host sends a CIG Control message with the new CIG state set to C\_SIMULATION or C\_MCC\_SIMUL.

The function call is **init\_simulation (state, curr\_sim\_buf, sim\_pdbase, pdynl, ptanks, pmodels, peffects)**, where:

*state* is the state the CIG is to be set up in (C\_DB\_SETUP or C\_MCC\_SETUP)  
*curr\_sim\_buf* is the double buffer currently being loaded by the simulation  
*sim\_pdbase* is a pointer to the primary database control block  
*pdynl* is a pointer to the structure containing the current state of all dynamic models  
*ptanks* is a pointer to the tank table  
*pmodels* is a pointer to the model table  
*peffects* is a pointer to the dynamic effects table

init\_simulation does the following:

- If AGPT mode is enabled, calls sim\_gmbox. AGPT mode is a non-standard version of the GT100 system.
- Initializes various variables.
- Creates a dummy system view flags message to initialize the array.
- Calls backend\_sim\_init to prepare the backend to run a simulation.
- Calls sim\_bal\_init to initialize variables used in Ballistics messages.
- Calls sim\_bal\_start to put Ballistics into the run state and tell it where active area memory is.
- Initializes the simulated vehicle's AGL (altitude above ground level).
- Zeroes the system frame counter.
- Initializes the static tank and model pointers.
- Initializes the multiple and long effects pointers.
- If a second AAM is present, initializes its static tank and model pointers, and multiple and long effects pointers.
- Initializes the static vehicle pointers to multiple-frame effects.
- Initializes the multiple-frame effects pointers to subroutine calls.
- Turns off the system view flags.
- If a second AAM is present, updates it also.
- Clears any stray messages from the SIMULATION\_MB mailbox.
- Posts a message and waits for the rowcol\_rd task(s) to perform database management (unless database updates were disabled on the startup command line).
- Posts a message and waits for local\_terrain to perform database feedback; sets the local terrain count to 0.
- Calls sysrup\_on to enable system interrupts.
- Calls sim\_bal\_frame\_rate to tell Ballistics the current CIG frame rate.
- Calls get\_dtp\_bank to determine which double buffer to use.
- Sets pointers to the dynamic model state table and the tank, model, and effects tables.
- If a second AAM is present, sets pointers to its tank, model, and effects tables.
- Calls \_pend\_on\_frame\_interrupt to wait for the next frame interrupt.

Called By: simulation

Routines Called:

_pend_on_frame_interrupt	
AAM2_ADDR	
backend_sim_init	
blcopy	
exit	
get_dtp_bank	
perror	(in AGPT mode only)
printf	
sc_accept	
sc_pend	
sc_post	
sim_bal_frame_rate	
sim_bal_init	
sim_bal_start	
sim_gmbox	(in AGPT mode only)
sysrup_on	

Parameters:	INT_2 INT_2 DB_INFO INT_4 TANK INT_4 INT_4	state *curr_sim_buf **sim_pdbase **pdynl **ptanks **pmodels **peffects
Returns:	none	

### 2.13.28 load\_dbase.c

The `load_dbase` function loads the terrain database into active area memory, and sets up various tables with the necessary data from the database. It also calls `open_ded` to load the contents of the dynamic elements database (DED). `load_dbase` is called after the viewport configuration tree has been created.

The function call is `load_dbase(pdbase)`, where *pdbase* is a pointer to the primary database control block.

`load_dbase` does the following:

- Determines how much generic memory is available. (This controls how many models can be read in.)
- If not enough memory is available, truncates the number of bytes to what is available.
- Reads in the data from the specified database.
- Reads in the overflow terrain data, if there is sufficient room.
- Calls `open_ded` to open the dynamic elements database, read the models in, and process them.
- Calls `load_modules` to load the initial load modules.
- Initializes the Load Module Branch Table, subroutine call table, and field-of-view test table for a 3500-meter or 7000-meter viewing range.

Called By: db\_mcc\_setup

Routines Called:

- bus\_error
- load\_modules
- open\_ded
- printf
- XLSEEK
- XREAD

Parameters:	DB_INFO	*pdbase
-------------	---------	---------

Returns:	none
----------	------

### 2.13.29 load\_esifa.c

The load\_esifa function downloads textures data to the ESIFA, for later download to the Pixel Processor Tilers (PPTs). This function is called when the backend is set up at system initialization time.

The function call is load\_esifa(pfilename, id), where:

*pfilename* is the name of the ESIFA textures file  
*id* is the backend id

load\_esifa does the following:

- Makes sure the esifa\_load\_wanted variable is TRUE.
- Opens the specified file.
- Reads the first line of the file to get the masks and file name need by esifa\_load.
- Calls esifa\_load to load the data into the subsystem boards.
- Closes the file.

Called By: backend\_setup

Routines Called: esifa\_load  
fclose  
fgets  
fopen  
printf  
scanf

Parameters: UNS\_1 \*pfilename[]  
INT\_4 id

Returns: none

### 2.13.30 load\_modules.c

The functions in the load\_modules.c CSU are used to load new active area rows and columns from the terrain database when required. These functions are:

- load\_modules
- getlmdp
- getside
- whatdirptr



### 2.13.30.1 load\_modules

The `load_modules` function loads a portion of the terrain database into AAM, when AAM needs to be completed loaded or reloaded. This function is called to read the initial load modules into AAM, and is called during a simulation if the simulated vehicle is detected to be out of viewing range of active area memory. (Specifically, the vehicle must be more than one-half the width of AAM outside its boundaries. In this instance, none of the terrain that is currently visible to the vehicle is in AAM — usually, this is due to "warping" across the terrain. The `_rowcol_rd` function then calls `load_modules` to rebuild all of AAM based on the vehicle's current location.)

The function call is `load_modules(pdbase)`, where *pdbase* is a pointer to the primary database control block.

`load_modules` does the following:

- Determines the viewing range.
- Calls `init_generic_lm` to initialize a generic load module for the applicable viewing range.
- Initializes the direction offsets.
- Calculates the southwest corner of AAM based on the current coordinates of the simulated vehicle.
- Calculates the four borders of AAM.
- Starting at the southernmost row and working up, reads each AAM row from west to east, calling `getside` to load the appropriate load modules from the database.
- Calls `whatdirptr` to find the direction pointer after the first row of load modules is loaded.
- After reaching the northernmost row, resets the address of the south border.

Called By:            `_rowcol_rd`  
                      `load_dbase`

Routines Called:    `getside`  
                      `GLOB`  
                      `init_generic_lm`  
                      `printf`  
                      `whatdirptr`

Parameters:           `DB_INFO`                                `*pdbase`

Returns:                `none`

### 2.13.30.2 getlmdp

The `getlmdp` function gets a load module's disk pointer from the database.

The function call is `getlmdp(xmod, ymod, pdbase)`, where:

*xmod* is the load module array number x  
*ymod* is the load module array number y  
*pdbase* is a pointer to the primary database control block

The function returns 0 if the load module is not in the database. If 0 is returned, *getside* (the calling function) calls *generic\_lm* to get a generic load module.

Called By:	<i>getside</i>		
Routines Called:	XLSEEK XREAD		
Parameters:	INT_4 INT_4 DB_INFO		<i>xmod</i> <i>ymod</i> * <i>pdbase</i>
Returns:	dbde.lm_loc 0		

### 2.13.30.3 *getside*

The *getside* function loads part of a row or column from the terrain database into active area memory. The number of load modules in the row or column actually loaded into AAM is 16 (the normal height/width of AAM) or 32 (the height/width of AAM if load module blocking is enabled).

The function call is *getside(lmdloc, xmod, ymod, xinc, yinc, diroff, pdbase, zeroit)*, where:

*lmdloc* is a pointer to the first load module on disk  
*xmod* is the first load module's array number x (west column)  
*ymod* is the first load module's array number y (south row)  
*xinc* is the load module's array number increment x  
*yinc* is the load module's array number increment y  
*diroff* is a byte offset to the directory pointer in the load module header  
*pdbase* is a pointer to the primary database control block  
*zeroit* is a flag used to determine when the running average load module centroid should be zeroed

*getside* does the following for each new load module it loads into AAM:

- Sets the load module number and gets its AAM address.
- Calls *getlmdp* to get the load module's disk pointer from the database.
- If the load module is not in the database, calls *generic\_lm* to get a generic load module.
- Reads in the load module.
- If working in subsystem 0, informs Ballistics of the new load module by pushing a MSG\_B0\_LM\_READ message onto the Ballistics message queue.

- Updates the field-of-view tables for the new load module.

Called By:           \_rowcol\_rd  
                  load\_modules

Routines Called:    getlmdp  
                      generic\_lm  
                      GLOB  
                      mx\_push  
                      XLSEEK  
                      XREAD

Parameters:	WORD	lmdloc
	INT_4	xmod
	INT_4	ymod
	INT_4	xinc
	INT_4	yinc
	WORD	diroff
	DB_INFO	*pdbase
	WORD	zeroit

Returns:           none

#### 2.13.30.4   whatdirptr

The whatdirptr function finds the direction pointer for the load module at a specified location in a specified direction.

The function call is **whatdirptr(xmod, ymod, diroff, pdbase)**, where:

*xmod* is the load module's array number x (west column)  
*ymod* is the load module's array number y (south row)  
*diroff* is the byte offset to the direction pointer in the load module header  
*pdbase* is a pointer to the primary database control block

Called By:           \_rowcol\_rd  
                  load\_modules

Routines Called:    none

Parameters:	INT_4	xmod
	INT_4	ymod
	WORD	diroff
	DB_INFO	*pdbase

Returns: <direction pointer>

### 2.13.31 loc\_ter.c

The loc\_ter.c CSU contains functions used to build local terrain messages for return to the Simulation Host. This feature is called "database feedback." These functions are:

- local\_terrain
- local\_terrain\_cleanup

A local terrain message describes the terrain, roads, rivers, and buildings that lie within the four grids surrounding the simulated vehicle. (One grid is usually 125 meters per side. One load module is defined as four grids: two rows by two columns.)

The first frame at which a local terrain message is created, and the frame interval at which new messages are generated, are defined in the memory\_map\_defines.h file. Currently, the first frame is set to 16 and the default interval is set to 32. The interval can be changed using the MSG\_DR11\_PKT\_SIZE or MSG\_LT\_STATE message.

Local terrain messages can also be sent on demand in response to a MSG\_RTN\_LT (return local terrain) message. This message is to be used by the MCC station only.

Each local terrain message contains the following:

- A header that specifies the number of polygon definitions and the number of bounding volumes (bvols) contained in the message.
- Polygons that describe the local terrain and the objects in it. These polygons are planar, convex, and three- or four-sided. Each polygon entry in the message specifies the soil type, priority code, minimum and maximum coordinates, and all polygon vertices in counter-clockwise order.
- Bounding volumes. A bvol definition contains one or more four-sided bounding boxes each of which has a planar, convex, polygonal base and a height (expressed in units on the z axis) for the volume given. Each bvol entry in the message specifies the bvol's height above the polygonal base, the bvol type identifier, the minimum and maximum coordinates, and the vertex list.

#### 2.13.31.1 local\_terrain

The local\_terrain function builds local terrain messages based on the simulated vehicle's current position. These messages are used by the Simulation Host to provide collision detection with objects in the simulated environment, and to calculate the dynamics of the vehicle in operation.

The local\_terrain task is loaded during the task initialization state. It is then suspended until a message is posted to its mailbox (LOCAL\_TERRAIN\_MB). local\_terrain assumes that the simulation vehicle's current position (stored in the viewport positions array) has just been updated.

local\_terrain does the following:

- Initializes the local terrain output buffer header (version and level).
- Calls `poll_shutdown` to see if a system shutdown has been requested.
- Waits for a message to be posted to the `LOCAL_TERRAIN_MB` mailbox.
- Calls `bal_get_db_pos` to find the simulated vehicle's current load module number and grid number.
- Calls `bal_get_lm_grid` to find the four grids that surround the simulated vehicle.
- Determines whether a new local terrain message needs to be built (i.e., if the simulated vehicle's position has changed since the last local terrain message).
- If the vehicle has moved, reinitializes the local terrain output buffer.
- Searches the four grids that lie around the simulated vehicle for polygons, and builds the polygon portion of the message.
- Searches the four grids that lie around the simulated vehicle for polygon components, and builds the polygon component portion of the message.
- Searches the four grids that lie around the simulated vehicle for bounding volumes, and builds the `bvol` portion of the message.
- Sets a pointer to the new local terrain message data.
- Creates the message header: message size, message type (`MSG_LOCAL_TERRAIN`), and the total number of polygons and `bvols` in the message. (The full message is divided into a series of smaller `MSG_LT_PIECE` messages by `loc_ter_msg` in the Message Processing CSC.)
- Posts a message to the `RTN_TERRAIN_MB` mailbox.

Called By.                    none    (task created and started at initialization)

Routines Called:            `bal_get_db_pos`  
                               `bal_get_lm_grid`  
                               `FXT0881`  
                               `GLOB`  
                               `poll_shutdown`  
                               `printf`            (in debug mode only)  
                               `read_watch`  
                               `rt_pend`  
                               `rt_post`

Parameters:                none

Returns:                    none

### 2.13.31.2 `local_terrain_cleanup`

The `local_terrain_cleanup` function deallocates the resources owned by the `local_terrain` task. This function is called if a system shutdown is requested by the Gossip user. The function is called via the `*task_cleanup` function pointer, which points to the cleanup routine's name in the task table.

The function call is `local_terrain_cleanup()`.

The function returns 1 if successful, or 0 if an error occurred.

**Note:** *This function is not yet implemented. At the current time, it simply returns a 1 if called.*

Called By: poll\_shutdown (through \*task\_cleanup)

Routines Called: none

Parameters: none

Returns: 0  
1

### 2.13.32 make\_bbn.c

The functions in the make\_bbn.c CSU can be used to make and modify hull-to-world matrices. These functions are:

- prt\_mtx
- rotate\_x
- rotate\_y
- rotate\_z
- multmatrix
- id\_matrix

**Note:** *These routines are called only by model\_mtx, which is not currently called by any other function. Therefore, these functions are not currently used.*

#### 2.13.32.1 prt\_mtx

The prt\_mtx function copies a matrix in memory.

The function call is **prt\_mtx(matrix, pntr)**, where:

*matrix* is the matrix

*pntr* is a pointer to the destination memory location

Called By: model\_mtx

Routines Called: none

Parameters: REAL\_4 matrix[4][3]  
REAL\_4 \*pntr

Returns: none

### 2.13.32.2 rotate\_x

The rotate\_x function rotates a matrix about the X axis. It creates an identity matrix, rotates it appropriately, then multiplies it by the original matrix.

The function call is **rotate\_x(theta, matrix)**, where:

*theta* is the angle of rotation  
*matrix* is the matrix to be rotated

Called By: model\_mtx

Routines Called: cos  
id\_matrix  
multmatrix  
sin  
toradians

Parameters: REAL\_4 theta  
REAL\_4 matrix[4][3]

Returns: none

### 2.13.32.3 rotate\_y

The rotate\_y function rotates a matrix about the Y axis. It creates an identity matrix, rotates it appropriately, then multiplies it by the original matrix.

The function call is **rotate\_y(theta, matrix)**, where:

*theta* is the angle of rotation  
*matrix* is the matrix to be rotated

Called By: model\_mtx

Routines Called: cos  
id\_matrix  
multmatrix  
sin  
toradians

Parameters: REAL\_4 theta  
REAL\_4 matrix[4][3]

Returns: none

#### 2.13.32.4 rotate\_z

The rotate\_z function rotates a matrix about the Z axis. It creates an identity matrix, rotates it appropriately, then multiplies it by the original matrix.

The function call is **rotate\_z(theta, matrix)**, where:

*theta* is the angle of rotation  
*matrix* is the matrix to be rotated

Called By: model\_mtx

Routines Called: cos  
id\_matrix  
multmatrix  
sin  
toradians

Parameters: REAL\_4 theta  
REAL\_4 matrix[4][3]

Returns: none

#### 2.13.32.5 multmatrix

The multmatrix function multiplies (concatenates) two matrices together. This function is used to multiply a matrix by a rotation matrix.

The function call is **multmatrix(matrix, matrix\_tmp)**, where:

*matrix* is the rotation matrix  
*matrix\_tmp* is the matrix to be rotated

Called By: rotate\_x  
rotate\_y  
rotate\_z

Routines Called: none

Parameters: REAL\_4 matrix[4][3]  
REAL\_4 matrix\_tmp[4][3]



Returns: none

### 2.13.32.6 id\_matrix

The `id_matrix` function creates an identity matrix (positioned at the origin) for use in rotating matrices.

The function call is `id_matrix(matrix)`, where *matrix* is the identity matrix to be created.

Called By: `model_mtx`  
`rotate_x`  
`rotate_y`  
`rotate_z`

Routines Called: none

Parameters: `REAL_4` `matrix[4][3]`

Returns: none

### 2.13.33 mkmtx\_nt.c

The functions in the `mkmtx_nt.c` CSU are used to perform computations on matrices. These functions are:

- `make_p_nt`
- `rotate_x_nt`
- `rotate_y_nt`
- `rotate_z_nt`
- `swap_axis`
- `id_4x3mtx`
- `scale_mtx`
- `translate`
- `mult_4x3mtx`
- `getmatrix`
- `matrix2`
- `mtxcpy`
- `mat_copy`
- `mat_vec_mul`
- `mat_adjugate`
- `mat_transpose`
- `mat_scale`
- `mat_determinant`
- `mat_inverse`
- `vec_mat_mul`
- `make4x3`

### 2.13.33.1 make\_p\_nt

The `make_p_nt` function converts a matrix to a perspective 4x3 matrix. The function creates an identity matrix, sets its field-of-view angles and offsets appropriately, then multiplies it by the original matrix.

The function call is `make_p_nt(itan_fov_i, itan_fov_j, hoffset_x, hoffset_y, matrix)`, where:

*itan\_fov\_i* is the inverse of the tangent of the horizontal field-of-view angle  
*itan\_fov\_j* is the inverse of the tangent of the vertical field-of-view angle  
*hoffset\_x* is the horizontal offset of the x coordinate  
*hoffset\_y* is the horizontal offset of the y coordinate  
*matrix* is the matrix to be converted

Called By:            `make_m1_overlays`  
                      `make_m2_overlays`

Routines Called:    `id_4x3mtx`  
                      `mult_4x3mtx`

Parameters:	<code>REAL_8</code>	<code>itan_fov_i</code>
	<code>REAL_8</code>	<code>itan_fov_j</code>
	<code>REAL_4</code>	<code>hoffset_x</code>
	<code>REAL_4</code>	<code>hoffset_y</code>
	<code>REAL_4</code>	<code>matrix[4][3]</code>

Returns:            none

### 2.13.33.2 rotate\_x\_nt

The `rotate_x_nt` function rotates a 4x3 matrix about the X axis. The function creates an identity matrix, rotates it accordingly, then multiplies it by the original matrix.

The function call is `rotate_x_nt(cos_theta, sin_theta, matrix)`, where:

*cos\_theta* is the cosine of the angle of rotation  
*sin\_theta* is the sine of the angle of rotation  
*matrix* is the matrix to be rotated

Called By:            `flea_init_cig_sw`  
                      `gos_model`

Routines Called:    `id_4x3mtx`  
                      `mult_4x3mtx`

Parameters:	REAL_8 REAL_8 REAL_4	cos_theta sin_theta matrix[4][3]
-------------	----------------------------	--

Returns:	none
----------	------

### 2.13.33.3 rotate\_y\_nt

The rotate\_y\_nt function rotates a 4x3 matrix about the Y axis. The function creates an identity matrix, rotates it accordingly, then multiplies it by the original matrix.

The function call is **rotate\_y\_nt(cos\_theta, sin\_theta, matrix)**, where:

*cos\_theta* is the cosine of the angle of rotation  
*sin\_theta* is the sine of the angle of rotation  
*matrix* is the matrix to be rotated

Called By:	flea_init_cig_sw gos_model
------------	-------------------------------

Routines Called:	id_4x3mtx mult_4x3mtx
------------------	--------------------------

Parameters:	REAL_8 REAL_8 REAL_4	cos_theta sin_theta matrix[4][3]
-------------	----------------------------	--

Returns:	none
----------	------

### 2.13.33.4 rotate\_z\_nt

The rotate\_z\_nt function rotates a 4x3 matrix about the Z axis. The function creates an identity matrix, rotates it accordingly, then multiplies it by the original matrix.

The function call is **rotate\_z\_nt(cos\_theta, sin\_theta, matrix)**, where:

*cos\_theta* is the cosine of the angle of rotation  
*sin\_theta* is the sine of the angle of rotation  
*matrix* is the matrix to be rotated

Called By:	flea_init_cig_sw flea_simulate_vehicles gos_model
------------	---

Routines Called:     id\_4x3mtx  
                      mult\_4x3mtx

Parameters:           REAL\_8                   cos\_theta  
                      REAL\_8                   sin\_theta  
                      REAL\_4                   matrix[4][3]

Returns:             none

### 2.13.33.5 swap\_axis

The swap\_axis function converts a matrix's axes so that the matrix conforms to the CIG's coordinate system, as follows:

```
xview = xworld
yview = -zworld
zview = yworld
```

The function call is **swap\_axis(matrix)**, where *matrix* is the matrix to be converted. swap\_axis first creates a 4x3 identity matrix, then sets this matrix to the following:

```
| 1  0  0 |
| 0  0  1 |
| 0 -1  0 |
| 0  0  0 |
```

swap\_axis then multiplies this matrix by the original matrix.

Called By:           make\_cal\_matrices  
                      make\_m1\_overlays  
                      make\_m2\_overlays

Routines Called:     id\_4x3mtx  
                      mult\_4x3mtx

Parameters:           REAL\_4                   matrix[4][3]

Returns:             none

### 2.13.33.6 id\_4x3mtx

The id\_4x3mtx function creates a 4x3 identity matrix (positioned at the origin) for use in rotating matrices. The identity matrix is as follows:

	1	0	0	
	0	1	0	
	0	0	1	
	0	0	0	

The function call is **id\_4x3mtx(matrix)**, where *matrix* is the new identity matrix.

Called By:           flea\_init\_cig\_sw  
                   flea\_simulate\_vehicles  
                   make\_cal\_matrices  
                   make\_m1\_overlays  
                   make\_m2\_overlays  
                   make\_p\_nt  
                   rotate\_x\_nt  
                   rotate\_y\_nt  
                   rotate\_z\_nt  
                   scale\_mtx  
                   swap\_axis  
                   translate

Routines Called:    none

Parameters:         REAL\_4                               matrix[4][3]

Returns:            none

### 2.13.33.7 scale\_mtx

The **scale\_mtx** function scales (enlarges, reduces, or skews) a 4x3 matrix. The function creates an identity matrix, applies the scaling factor to it, then multiplies it by the original matrix.

The function call is **scale\_mtx(scale, matrix)**, where:

*scale* is the scaling factor  
*matrix* is the matrix to be scaled

This function is not currently used. (Scaling is a future system enhancement.)

Called By:           none

Routines Called:    id\_4x3mtx  
                       mult\_4x3mtx

Parameters:         REAL\_4                               matrix[4][3]  
                       REAL\_4                               scale[3]

Returns: none

### 2.13.33.8 translate

The translate function moves a matrix to a new position by adding a translation value to each of its coordinates. The function creates an identity matrix, translates its coordinates, then multiplies it by the original matrix.

The function call is **translate(xval, yval, zval, matrix)**, where:

*xval* is the amount to be added to the x coordinate  
*yval* is the amount to be added to the y coordinate  
*zval* is the amount to be added to the z coordinate  
*matrix* is the matrix to be translated

Translation amounts are specified in meters.

Called By: flea\_init\_cig\_sw  
model\_mtx

Routines Called: id\_4x3mtx  
mult\_4x3mtx

Parameters: REAL\_4 xval  
REAL\_4 yval  
REAL\_4 zval  
REAL\_4 matrix[4][3]

Returns: none

### 2.13.33.9 mult\_4x3mtx

The mult\_4x3mtx function multiplies two 4x3 matrices together. This function is used to multiply a matrix by a rotation matrix.

The function call is **mult\_4x3mtx(matrix, matrix\_tmp)**, where:

*matrix* is the rotation matrix  
*matrix\_tmp* is the matrix to be rotated

Called By: make\_p\_nt  
rotate\_x\_nt  
rotate\_y\_nt  
rotate\_z\_nt  
scale\_mtx

swap\_axis  
translate

Routines Called: none

Parameters: REAL\_4 matrix[4][3]  
REAL\_4 matrix\_tmp[4][3]

Returns: none

#### 2.13.33.10 getmatrix

The getmatrix function concatenates a matrix with another matrix.

The function call is **getmatrix(matrix, matrix\_tmp)**, where:

*matrix* is the original matrix and the result matrix

*matrix\_tmp* is the matrix to concatenate with the original matrix

This function is not currently used.

Called By: none

Routines Called: none

Parameters: REAL\_4 matrix[4][3]  
REAL\_4 matrix\_tmp[4][3]

Returns: none

#### 2.13.33.11 matrix2

The matrix2 function concatenates (multiplies) two matrices to create a third matrix.

The function call is **matrix2(matrixa, matrixb, matrixc)**, where:

*matrixa* and *matrixb* are the matrices to be concatenated

*matrixc* is the result

This function is not currently used.

Called By: none

Routines Called: none

Parameters:	REAL_4	matrixa [4][3]
	REAL_4	matrixb [4][3]
	REAL_4	matrixc [4][3]

Returns: none

### 2.13.33.12 mtxcpy

The mtxcpy function copies a matrix from one memory location to another.

The function call is **mtxcpy(to\_matrix, from\_matrix, matrix\_type)**, where:

*to\_matrix* is the destination location

*from\_matrix* is the source location

*matrix\_type* is the type of matrix (RTS3x3\_TYPE, RTS4x3\_TYPE, or ROT2x1\_TYPE)

Called By: concat\_mtx  
vpt\_cnode\_set\_matrix  
vpt\_update\_mtx

Routines Called: none

Parameters:	I4P	to_matrix
	I4P	from_matrix
	BYTE	matrix_type

Returns: none

### 2.13.33.13 mat\_copy

The mat\_copy function copies a T matrix from one memory location to another.

The function call is **mat\_copy(src, dest)**, where:

*src* is the source location

*dest* is the destination location

Called By: mat\_transpose

Routines Called: none



Parameters:	T_MATRIX T_MATRIX	src dest
-------------	----------------------	-------------

Returns:	none
----------	------

#### 2.13.33.14 mat\_vec\_mul

The `mat_vec_mul` function multiplies a T vector by a T matrix and stores the result.

The function call is `mat_vec_mul(m, v, result)`, where:

*m* is the matrix  
*v* is the vector  
*result* is the result

This function is not currently used.

Called By:	none
------------	------

Routines Called:	none
------------------	------

Parameters:	T_MATRIX T_VECTOR T_VECTOR	m v result
-------------	----------------------------------	------------------

Returns:	none
----------	------

#### 2.13.33.15 mat\_adjugate

The `mat_adjugate` function calculates the adjugate matrix for a given T matrix.

The function call is `mat_adjugate(m, result)`, where:

*m* is the original matrix  
*result* is the adjugate matrix

Called By:	mat_inverse
------------	-------------

Routines Called:	none
------------------	------

Parameters:	T_MATRIX T_MATRIX	m result
-------------	----------------------	-------------

Returns: none

#### 2.13.33.16 mat\_transpose

The `mat_transpose` function transposes a T matrix into a new matrix.

The function call is `mat_transpose(m, result)`, where:

*m* is the original matrix  
*result* is the transposed matrix

Called By: `mat_inverse`

Routines Called: `mat_copy`

Parameters: `T_MATRIX` `m`  
`T_MATRIX` `result`

Returns: none

#### 2.13.33.17 mat\_scale

The `mat_scale` function scales (enlarges, reduces, or skews) a T matrix.

The function call is `mat_scale(mat, scale_factor, result)`, where:

*mat* is the original matrix  
*scale\_factor* is the amount by which to scale  
*result* is the scaled matrix

Called By: `mat_inverse`

Routines Called: none

Parameters: `T_MATRIX` `mat`  
`REAL_8` `scale_factor`  
`T_MATRIX` `result`

Returns: none

### 2.13.33.18 mat\_determinant

The `mat_determinant` function returns the determinant of a T matrix.

The function call is `mat_determinant(m)`, where *m* is the original matrix.

Called By: `mat_inverse`

Routines Called: `none`

Parameters: `T_MATRIX` `m`

Returns: `<matrix determinant>`

### 2.13.33.19 mat\_inverse

The `mat_inverse` function calculates the inverse of a given T matrix.

The function call is `mat_inverse(m, result)`, where:

*m* is the original matrix  
*result* is the inverse matrix

To calculate the inverse of the matrix, `mat_inverse` does the following:

1. Gets the adjugate matrix.
2. Transposes the adjugate matrix.
3. Calculates the inverse of the determinant of the transposed matrix.
4. Scales the result of step 2 by the result of step 3.
5. Stores the result of step 4 in *result*.

This function is not currently used.

Called By: `none`

Routines Called: `mat_adjugate`  
`mat_determinant`  
`mat_scale`  
`mat_transpose`

Parameters: `T_MATRIX` `m`  
`T_MATRIX` `result`

Returns: none

### 2.13.33.20 vec\_mat\_mul

The `vec_mat_mul` function multiplies a T vector by a T matrix and stores the result.

The function call is `vec_mat_mul(v, m, result)`, where:

*v* is the vector  
*m* is the original matrix  
*result* is the new matrix

Called By: `find_pitch_and_roll`

Routines Called: none

Parameters:	T_VECTOR	v
	T_MATRIX	m
	T_VECTOR	result

Returns: none

### 2.13.33.21 make4x3

The `make4x3` function creates a 4x3 world/hull matrix given its heading, pitch, roll, and x, y, and z coordinates. It first computes values for the basic 3x3 portion of the matrix, then computes values for the x, y, z portion of the matrix.

The function call is `make4x3(cz, sz, cx, sx, cy, sy, x, y, z, matrix)`, where:

*cz* is the cosine of the z angle  
*sz* is the sine of the z angle  
*cx* is the cosine of the x angle  
*sx* is the sine of the x angle  
*cy* is the cosine of the y angle  
*sy* is the sine of the y angle  
*x* is the x coordinate  
*y* is the y coordinate  
*z* is the z coordinate  
*matrix* is the result matrix

Called By: `concat_mtx`  
`flea_simulate_vehicles`  
`flea_update_pos`

Routines Called: none

Parameters:	REAL_4	CZ
	REAL_4	SZ
	REAL_4	CX
	REAL_4	SX
	REAL_4	CY
	REAL_4	SY
	REAL_4	X
	REAL_4	Y
	REAL_4	Z
	T_MATRIX	matrix

Returns: none

### 2.13.34 model\_mtx.c

The model\_mtx function builds hull-to-world, turret-to-hull, and gun-to-turret matrices. The function call is **model\_mtx(modnum)**, where *modnum* is the model number.

This function is not currently used.

Called By: none

Routines Called: GLOB  
id\_matrix  
prt\_mtx  
rotate\_x  
rotate\_y  
rotate\_z  
translate

Parameters:	INT_2	modnum
-------------	-------	--------

Returns: none

### 2.13.35 mx2\_hword.c

The functions in the mx2\_hword.c CSU handle half-word messages exchanges. These function are used for the interface between the real-time software (specifically, the MPV Interface CSC routines) and the Force board. These functions are:

- mx2\_open
- mx2\_push
- mx2\_peek
- mx2\_skip
- mx2\_error
- mx2\_hwcopy

These functions are also present in and are used by the Force Processing CSC (see section 2.18). The Routines Called and Called By lists in this section do not include any Force Processing functions. Similarly, the Routines Called and Called By lists in section 2.18 do not include any real-time software functions.

### 2.13.35.1 mx2\_open

The `mx2_open` function initializes the message buffer given its start address and its size. At the current time, this function is used only by the Force Processing CSC (see section 2.18).

The function call is `mx2_open(dev_P, device_size)`, where:

*dev\_P* is a pointer to the message buffer  
*device\_size* is the size of the message buffer

The function always returns `MX_DEVICE_OPENED`.

Called By:	none	
Routines Called:	sc_lock sc_unlock	
Parameters:	MX2_DEVICE INT_4	*dev_P device_size
Returns:	MX_DEVICE_OPENED	

### 2.13.35.2 mx2\_push

The `mx2_push` function pushes a message onto the message buffer. `mx2_push` first checks to see if the buffer has room for the message, then calls `mx2_hwcopy` to copy the message to the buffer.

The function call is `mx2_push(dev_P, source_address, message_code, message_size)`, where:

*dev\_P* is a pointer to the message buffer  
*source\_address* is the address of the message  
*message\_code* is the message type indicator  
*message\_size* is the size of the message in bytes

The function returns `MX_MESSAGE_PUSHED` if successful. It returns `MX_DEVICE_FULL` if the message buffer is already full.

Called By:	mpvideo_boot
------------	--------------

mpvideo\_laser\_request\_range  
 mpvideo\_load  
 mpvideo\_send\_req  
 mpvideo\_set\_lut  
 mpvideo\_set\_mode  
 mpvideo\_set\_video  
 mpvideo\_sim\_init

Routines Called:   mx2\_hwcopy  
                       sc\_lock  
                       sc\_unlock

Parameters:	MX2_DEVICE	*dev_P
	WORD	source_address
	WORD	message_code
	WORD	message_size

Returns:           MX\_DEVICE\_FULL  
                       MX\_MESSAGE\_PUSHED

### 2.13.35.3   mx2\_peek

The `mx2_peek` function previews the top message in the message buffer and returns a pointer to it.

The function call is `mx2_peek(dev_P, message_code, message_size, message_addr)`, where:

*dev\_P* is a pointer to the message buffer  
*message\_code* is a pointer to the message type indicator  
*message\_size* is a pointer to the message size in bytes  
*message\_addr* is a pointer to the message address

The function returns `MX_MESSAGE_PREVIEWED` if successful. It returns `MX_DEVICE_EMPTY` if the message buffer contains no messages.

Called By:           mpvideo\_boot  
                       mpvideo\_load  
                       mpvideo\_response

Routines Called:   sc\_lock  
                       sc\_unlock

Parameters:	MX2_DEVICE	*dev_P
	WORD	*message_code
	WORD	*message_size
	BYTE	**message_addr

Returns:           MX\_DEVICE\_EMPTY  
                  MX\_MESSAGE\_PREVIEWED

#### 2.13.35.4   mx2\_skip

The mx2\_skip function skips over first message in the message buffer. The skipped message is flushed from the buffer and the next message moves to the head of the queue.

The function call is **mx2\_skip(dev\_P)**, where *dev\_P* is a pointer to the message buffer.

The function returns MX\_MESSAGE\_SKIPPED if successful. It returns MX\_DEVICE\_EMPTY if the message buffer contains no messages.

Called By:           mpvideo\_boot  
                  mpvideo\_load  
                  mpvideo\_response

Routines Called:    sc\_lock  
                  sc\_unlock

Parameters:          MX2\_DEVICE                               \*dev\_P

Returns:            MX\_MESSAGE\_SKIPPED  
                  MX\_DEVICE\_EMPTY

#### 2.13.35.5   mx2\_error

The mx2\_error function returns an ASCII translation of an error code, for output to the operator.

The function call is **mx2\_error(status)**, where *status* is the error condition.

This function is not currently used.

Called By:           none

Routines Called:    none

Parameters:          WORD                                       status

Returns:            "DEVICE CLOSED"  
                  "DEVICE TABLE FULL"



"DEVICE OPENED"  
 "DEVICE BUSY"  
 "DEVICE EMPTY"  
 "DEVICE FULL"  
 "MESSAGE PUSHED"  
 "MESSAGE POPPED"  
 "MESSAGE PREVIEWED"  
 "MESSAGE SKIPPED"  
 "UNDEFINED ERROR"  
 "UNDEFINED RETURN"

### 2.13.35.6 mx2\_hwcopy

The mx2\_hwcopy function does a half-word block copy.

The function call is **mx2\_hwcopy(source\_P, destination\_P, byte\_count)**, where:

*source\_P* is the source location  
*destination\_P* is the destination location  
*byte\_count* is the number of bytes to be copied

Called By: mpvideo\_load  
 mpvideo\_response  
 mx2\_push

Routines Called: none

Parameters: INT\_2 \*source\_P  
 INT\_2 \*destination\_P  
 INT\_2 byte\_count

Returns: none

### 2.13.36 open\_dbase.c

The functions in the open\_dbase.c CSU are used to open the terrain database and initialize parameters for Ballistics. These functions are:

- open\_dbase
- func\_msg

#### 2.13.36.1 open\_dbase

The open\_dbase function opens the terrain database and initializes configuration and active area memory parameters for Ballistics. This function is called when the Simulation Host sends a MSG\_FILE\_DESCR - DB\_SETUP message.

The function call is **open\_dbase(pdbase)**, where *pdbase* is a pointer to the primary database control block.

**open\_dbase** does the following:

- Closes any open database.
- Opens the database file specified in the message or entered through the keyboard; calls **find\_fn** to find the latest version of the specified file.
- Initializes the database header data to zero.
- Reads the file header.
- Allocates memory for a work area.
- Reads the tag record and reads the applicable data.
- Frees the allocated memory.
- Allocates memory for the terrain model address and catalog tables.
- Allocates memory for the special effects address and catalog tables.
- Initializes database variables: number of load module blocks per side, grid space, and number of load modules on a side of AAM.
- Takes the AAM address from the terrain database header and ORs it with the current terrain\_addr which is set up with the AAM offset (from config\_database).
- Copies the terrain\_addr into the field-of-view table address. For a 7000-meter viewing range (i.e., with load module blocking enabled), divides this address by 2.
- Sets the remaining database variables: number of load modules per side of a load module block, load module width, load module block width, active area width, total number of load modules and load module blocks, etc.
- Calculates the amount of memory required for the database and calls **extended\_ram\_available** to see if the database is going to fit in the available RAM.
- Initializes Ballistics configuration parameters: processor type, frame rate, number of AAM partitions, maximum chord length, maximum model radius, maximum number of models, maximum number of active rounds, polygons, and bvols, etc.
- Sends the configuration data to Ballistics by pushing a **MSG\_B0\_BAL\_CONFIG** message onto the Ballistics message queue.
- Initializes AAM partition information for Ballistics: number of load modules per side, total number of load modules in AAM, viewing distance, grid width, AAM base address, etc.
- Sends the AAM partition parameters to Ballistics by pushing a **MSG\_B0\_DATABASE\_INFO** message onto the Ballistics message queue.

The terrain database is loaded into active area memory by **load\_dbase**, which is called by **db\_mcc\_setup** after the viewport configuration tree is created.

Called By:            **db\_mcc\_setup**

Routines Called:    **exit**  
                      **extended\_ram\_available**  
                      **find\_fn**  
                      **free**  
                      **func\_msg**  
                      **GLOB**  
                      **malloc**  
                      **mx\_push**  
                      **printf**  
                      **strlen**

SYSERR  
XCLOSE  
XLSEEK  
XOPEN  
XREAD

Parameters: DB\_INFO \*pdbase

Returns: none

### 2.13.36.2 func\_msg

The `func_msg` function outputs error messages related to opening the database. Specifically, it outputs "OPEN\_DBASE" followed by text supplied by `open_dbase`.

The function call is `func_msg(s)`, where *s* is the additional text to be output to stdout.

Called By: open\_dbase

Routines Called: printf

Parameters: BYTE s[]

Returns: none

### 2.13.37 open\_ded.c

The `open_ded` function opens the dynamic elements database (DED) file and processes the dynamic model list, changing the relative AAM addresses to absolute AAM addresses. The DED file name can be specified by the Simulation Host in the MSG\_FILE\_DESCR - DB\_DED\_SETUP\_0 or DB\_DED\_SETUP\_1 message, or it can be entered through the keyboard.

For 7000-meter databases, additional memory can be used to load extra DED models. The memory is allocated in two areas called pools: pool 0 is the standard pool and pool 1 is the additional ("wrap") pool. `open_ded` calls the `pool_init` function to initialize the standard pool and allocate the wrap pool if the additional memory is available and required.

The function call is `open_ded (pdbase, avail_gm, vme_offset)`, where:

*pdbase* is a pointer to the primary database control block  
*avail\_gm* is the amount of space in generic memory for model information  
*vme\_offset* is the VME address offset to active area memory

`open_ded` is called after the terrain database has been loaded into active area memory. `open_ded` does the following:

- Closes any open DED file.
- Allocates memory for a scratch buffer.
- Finds and opens the latest version of the specified DED file.
- Reads the database header and verifies it is valid.
- Allocates memory for the DED model address and catalog tables.
- Allocates memory for the DED special effects address and catalog tables.
- Initializes the file\_cut (the point at which the DED file will be split into two pools, if required).
- Initializes the address of the first model to be placed into the wrap pool to 0.
- Reads in the model directory entries, and updates the address of the first model to go into the wrap pool.
- Reads in the effects directory, and updates the first wrap address again.
- Sets the file\_cut value:
  - If the DED is too large for standard generic memory, the file\_cut is the first wrap address (i.e., a model boundary).
  - If the DED fits into standard generic memory, the file\_cut is the size of the standard pool.
- Sets up the DED memory pools:
  - Calls pool\_init to initialize the pool(s).
  - Outputs the pool boundaries to stdout.
- Determines the memory address to shift boundary data to (the last allocated pool's start address plus the size of DED).
- Loads all models into generic model active area memory.
- Calls download\_bvols to download the models and bounding volumes to Ballistics.
- Processes the model directory entries; calls ded\_model\_trace to trace all DTP commands.
- Processes the special effect directory entries; calls ded\_model\_trace to trace all DTP commands.
- Closes the DED file.
- Sets all invalid model and effect address table entries to the default entry. An address is considered invalid if it is 0 or is outside of both pools.
- Frees the scratch buffer.

The function returns 0 if the DED file is fully or partially loaded. It returns -1 if no DED file is found or the database header is invalid.

Called By:           load\_dbase

Routines Called:    addr\_in\_pool  
                      ded\_model\_trace  
                      download\_bvols  
                      find\_fn  
                      free  
                      malloc  
                      pool\_init  
                      printf  
                      strlen  
                      XCLOSE  
                      XLSEEK  
                      XOPEN  
                      XREAD

Parameters:	DB_INFO INT_4 WORD	*pdbase; avail_gm vme_offset
Returns:	0 -1	

### 2.13.38 otherveh\_state.c

The `otherveh_state` function performs other vehicle model control. This function is called whenever the Simulation Host sends a `MSG_OTHERVEH_STATE` message to reposition dynamic vehicles in active area memory.

The function call is `otherveh_state(msgp, pdbase, vme_offset, db, model_num)`, where:

*msgp* is a pointer to the `MSG_OTHERVEH_STATE` message  
*pdbase* is a pointer to the primary database control block  
*vme\_offset* is the VME address offset to active area memory  
*db* is the current double-buffer base pointer  
*model\_num* is 1 (first model this frame) or 0 (not the first model)

`otherveh_state` does the following:

- If *model\_num* is 1, resets the dynamic component counters.
- Checks to see if the vehicle is a tank.
  - If it is a tank, sets `veh_is_tank` to TRUE; makes sure there is enough room for a new tank; adds the tank.
  - If it is not a tank, sets `veh_is_tank` to FALSE; makes sure there is room for a new vehicle; adds the vehicle.
- Saves the model's matrix and center.
- Adds the model to the proper load module.
- Determines which list to add the model to (pre-process, post-process, or standard).

The function returns 0 if successful. It returns `E_LIMIT` if there is no room for a new model.

Called By: `msg_otherveh_state`

Routines Called: `FIND_LM`

Parameters:	MSG_OTHERVEH_STATE DB_INFO INT_4 INT_4 INT_4	*msgp *pdbase vme_offset db model_num
-------------	--	---

Returns:           0  
                  E\_LIMIT

### 2.13.39     pretend\_veh.c

The `pretend_veh` function processes Gossip vehicles. This function is called at the end of every frame to process changes to vehicles that were input by the Gossip user (via the Model menu) during the previous frame. `pretend_veh` adds and removes static vehicles, adds effects, and adds dynamic vehicles.

The function call is `pretend_veh (imsg, pretend_sv, pretend_ef, pretend_dv, remove_sv, INBUF, frame_count )`, where:

*imsg* is a pointer to the message packet from the Simulation Host  
*pretend\_sv* is a pointer to the static vehicle to be added  
*pretend\_ef* is a pointer to the effect to be added  
*pretend\_dv* is a pointer to the dynamic vehicle to be added  
*remove\_sv* is a pointer to the static vehicle to be removed  
*INBUF* is a pointer to the incoming (SIM-to-CIG) message buffer  
*frame\_count* is the current frame number

The changes to the Gossip vehicles are entered via menus. The `gos_model` function builds Simulation Host-type messages for the changes and places them in the locations specified in the call (`pretend_sv`, `pretend_dv`, etc.). `pretend_veh` then takes the messages and puts them in the next incoming message packet for processing.

Requests to remove static vehicles may also come from the `_rowcol_rd` function. `_rowcol_rd` periodically checks to see that all static vehicles are within viewing range of the simulated vehicle. If a vehicle is detected to be out of range, `_rowcol_rd` generates a message to delete it and places it in the `remove_sv` location, for processing the next frame by `pretend_veh`.

`pretend_veh` returns TRUE if successful, or FALSE if it cannot process the message.

Called By:           process\_a\_msg

Routines Called:    getenv  
                      printf

Parameters:	INT_4	**imsg
	INT_4	**pretend_sv
	INT_4	**pretend_ef
	INT_4	**pretend_dv
	INT_4	**remove_sv
	MSG5_BLK	*INBUF
	INT_4	frame_count

Returns:            1 (TRUE)

0 (FALSE)

#### 2.13.40 rowcol\_rd.c

The functions in the rowcol\_rd.c CSU are responsible for determining whether new load modules need to be read from the terrain database into active area memory for hardware, local terrain, and Ballistics use. These functions are:

- rowcol\_rd\_1
- rowcol\_rd\_2
- rowcol\_rd\_3
- rowcol\_rd\_4
- \_rowcol\_rd
- rowcol\_rd\_1\_cleanup
- rowcol\_rd\_2\_cleanup
- rowcol\_rd\_3\_cleanup
- rowcol\_rd\_4\_cleanup
- \_rowcol\_rd\_cleanup

The \_rowcol\_rd function performs all database management, and the \_rowcol\_rd\_cleanup function performs all resource deallocation at shutdown. The other functions simply call \_rowcol\_rd or \_rowcol\_rd\_cleanup for a specified subsystem/active area memory.

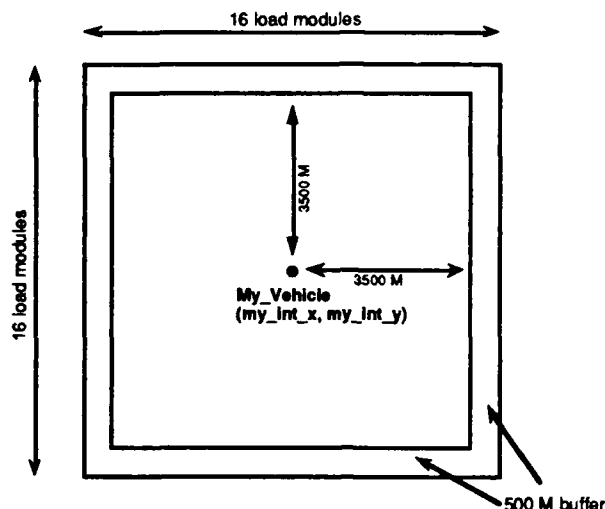
The terrain database, which is stored in the CIG, describes the entire terrain that can be displayed in the simulation. It also contains the graphic information used to display vehicles, houses, trees, hills, and other objects in the terrain.

The items stored in the terrain database are represented by connected polygons that are three-dimensional images. The polygons are grouped into compacted data structures such as terrain grids, polygon models, and stamp arrays. These structures are further grouped into unique static objects (rivers, roads, and other features that appear only once in the database) and generic models (houses, trees, vehicles, and other features that commonly recur in the database).

The terrain database is divided into units called load modules. One load module contains the instructions and data required to process a one-half kilometer square area of static objects. Each load module contains all the roads, rivers, terrains, buildings, and other features within a 500 by 500 meter area. The load modules in the terrain database are organized in rows and columns. The total size of the database is variable.

Each load module is divided into four areas called grids. Each grid is a 125-meter by 125-meter square.

Active area memory (AAM) contains the subset of the local terrain that can be viewed and interacted with at a given point in time by the simulation. The AAM stores an 8K by 8K area centered around the simulation vehicle. This provides a viewing range of 3500 meters in each direction, with a 500-meter buffer along each edge. The AAM contains 256 load modules (16 rows by 16 columns).



As the simulated vehicle moves toward an edge of active area memory, `_rowcol_rd` brings in new load modules from the terrain database, overwriting those areas that the vehicle is moving away from. The objective of this process is to keep the simulated vehicle in the center of active area memory.

Active area memory can be thought of as a window that moves over the terrain database. As the vehicle travels east, for example, the window must be moved east to keep the vehicle in the window's center. To do this, `_rowcol_rd` determines what column in the database lies east of the current east boundary of AAM. It then reads part of that column (the 16 load modules in the column that lie between AAM's north and south boundaries) into AAM. Finally, it shifts the west boundary of AAM over one column.

With very large terrain databases, load module blocking can be enabled. One load module block contains four load modules (two rows by two columns). Therefore, one load module block is 1000 meters by 1000 meters, or a one-kilometer square area. Load module blocking increases the amount of terrain that can be loaded into active area memory to 16K by 16K. It also doubles the viewing range of the simulated vehicle (from 3500 meters to 7000 meters).

#### 2.13.40.1 `rowcol_rd_1`

The `rowcol_rd_1` function calls `_rowcol_rd` for subsystem 0. `rowcol_rd_1` is invoked by a mailbox message posted by `init_simulation`.

The function call is `rowcol_rd_1()`.

Called By: none (task created at initialization time)

Routines Called: `_rowcol_rd`

Parameters: none



Returns: none

#### 2.13.40.2 rowcol\_rd\_2

The rowcol\_rd\_2 function calls \_rowcol\_rd for subsystem 1. rowcol\_rd\_2 is invoked by a mailbox message posted by init\_simulation, if subsystem 1 is defined.

The function call is rowcol\_rd\_2().

Called By: none (task created at initialization time)

Routines Called: \_rowcol\_rd

Parameters: none

Returns: none

#### 2.13.40.3 rowcol\_rd\_3

The rowcol\_rd\_3 function calls \_rowcol\_rd for subsystem 2. rowcol\_rd\_3 is invoked by a mailbox message posted by init\_simulation, if subsystem 2 is defined.

The function call is rowcol\_rd\_3().

Called By: none (task created at initialization time)

Routines Called: \_rowcol\_rd

Parameters: none

Returns: none

#### 2.13.40.4 rowcol\_rd\_4

The rowcol\_rd\_4 function calls \_rowcol\_rd for subsystem 3. rowcol\_rd\_4 is invoked by a mailbox message posted by init\_simulation, if subsystem 3 is defined.

The function call is rowcol\_rd\_4().

Called By: none (task created at initialization time)

Routines Called:     \_rowcol\_rd

Parameters:           none

Returns:              none

#### 2.13.40.5   \_rowcol\_rd

The \_rowcol\_rd function determines whether a new row or column of the terrain database needs to be read into active area memory. This task is invoked by the rowcol\_rd\_\* functions when a message is posted to the database management mailbox. The primary function of \_rowcol\_rd is to keep the simulated vehicle in the center of active area memory.

The function call is **\_rowcol\_rd(pdbase)**, where *pdbase* is a pointer to the primary database control block.

When a message is posted to its mailbox, \_rowcol\_rd does the following:

- Initializes direction offsets.
- Calls poll\_shutdown to see if a system shutdown has been requested.
- Checks to see if reading\_lm is TRUE, indicating that load modules were read in during the last iteration:
  - If TRUE, sets it to FALSE.
  - If FALSE, posts a messages and waits to be woken up again.
- Pushes a MSG\_B0\_AAM\_SW\_CORNER message onto the Ballistics message queue to tell Ballistics where the southwest corner of active area memory is.
- Checks to see if the simulated vehicle is out of viewing range of AAM (i.e., is beyond an AAM boundary by a distance of more than one-half AAM width). If yes:
  - Sets warp\_in\_progress and reading\_lm to TRUE.
  - Calls load\_modules to reload all of AAM from the terrain database.
  - Sets warp\_in\_progress to FALSE.
- Checks to see if the simulated vehicle is inside AAM, or outside but within viewing range of it. If yes:
  - Compares the coordinates of the vehicle's centroid to the center of AAM.
  - If the vehicle is detected to be off-center, sets reading\_lm to TRUE, then calls whatdirptr and getside to load a new row or column in the needed direction. For example, if the vehicle is detected to be too far away from the west boundary (i.e., is east of AAM center), a column is added to the east side and deleted from the west. This has the effect of shifting all of AAM east by one column.
  - Updates the applicable database data variables to reflect the change to AAM boundaries.
- Checks to make sure that all static vehicles are within the active area. If any static vehicle is found outside the simulated vehicle's viewing range:
  - Generates a MSG\_STATICVEH\_REM message (with the vehicle type set to 0).
  - Puts the message in global memory for processing (by pretend\_veh) during the next frame.

Called By:           rowcol\_rd\_1  
                      rowcol\_rd\_2  
                      rowcol\_rd\_3  
                      rowcol\_rd\_4

Routines Called:    getside  
                      GLOB  
                      load\_modules  
                      mx\_push  
                      poll\_shutdown  
                      printf  
                      rt\_pend  
                      rt\_post  
                      sc\_delay  
                      whatdirptr

Parameters:           DB\_INFO                               \*pdbase

Returns:             none

#### 2.13.40.6 rowcol\_rd\_1\_cleanup

The rowcol\_rd\_1\_cleanup function calls \_rowcol\_rd\_cleanup for subsystem 0. This function is called when a system shutdown is requested by the Gossip user. The function is called via the \*task\_cleanup function pointer, which points to the cleanup routine's name in the task table.

The function call is rowcol\_rd\_1\_cleanup().

Called By:           poll\_shutdown           (through \*task\_cleanup)

Routines Called:    \_rowcol\_rd\_cleanup

Parameters:           none

Returns:             \_rowcol\_rd\_cleanup( &db\_control[0] )

#### 2.13.40.7 rowcol\_rd\_2\_cleanup

The rowcol\_rd\_2\_cleanup function calls \_rowcol\_rd\_cleanup for subsystem 1. This function is called when a system shutdown is requested by the Gossip user. The function is called via the \*task\_cleanup function pointer, which points to the cleanup routine's name in the task table.

The function call is **rowcol\_rd\_2\_cleanup()**.

Called By: poll\_shutdown (through \*task\_cleanup)

Routines Called: \_rowcol\_rd\_cleanup

Parameters: none

Returns: \_rowcol\_rd\_cleanup( &db\_control[1] )

#### 2.13.40.8 rowcol\_rd\_3\_cleanup

The rowcol\_rd\_3\_cleanup function calls \_rowcol\_rd\_cleanup for subsystem 2. This function is called when a system shutdown is requested by the Gossip user. The function is called via the \*task\_cleanup function pointer, which points to the cleanup routine's name in the task table.

The function call is **rowcol\_rd\_3\_cleanup()**.

Called By: poll\_shutdown (through \*task\_cleanup)

Routines Called: \_rowcol\_rd\_cleanup

Parameters: none

Returns: \_rowcol\_rd\_cleanup( &db\_control[2] )

#### 2.13.40.9 rowcol\_rd\_4\_cleanup

The rowcol\_rd\_4\_cleanup function calls \_rowcol\_rd\_cleanup for subsystem 3. This function is called when a system shutdown is requested by the Gossip user. The function is called via the \*task\_cleanup function pointer, which points to the cleanup routine's name in the task table.

The function call is **rowcol\_rd\_4\_cleanup()**.

Called By: poll\_shutdown (through \*task\_cleanup)

Routines Called: \_rowcol\_rd\_cleanup

Parameters: none

Returns: `_rowcol_rd_cleanup( &db_control[3] )`

#### 2.13.40.10 `_rowcol_rd_cleanup`

The `_rowcol_rd_cleanup` function deallocates the resources allocated to the `rowcol_rd` task for a specified backend. This function is called when a system shutdown is requested by the Gossip user.

The function call is `rowcol_rd_cleanup(pdbase)`, where *pdbase* is the primary database control block for the backend.

The function returns 1 if successful, or 0 if an error occurred.

*Note: This function is not yet implemented. At the current time, the function simply returns a 1 if called.*

Called By: `rowcol_rd_1_cleanup`  
`rowcol_rd_2_cleanup`  
`rowcol_rd_3_cleanup`  
`rowcol_rd_4_cleanup`

Routines Called: none

Parameters: `DB_INFO` `*pdbase`

Returns: 0  
1

#### 2.13.41 `rt_mailbox.c`

The `rt_mailbox.c` CSU contains special intertask mailbox functions used by the real-time software tasks.

The functions in this CSU are:

- `rt_pend`
- `rt_post`
- `rt_accept`

##### 2.13.41.1 `rt_pend`

The `rt_pend` function waits for a message to be posted to a specific mailbox.

The function call is `rt_pend(mboxp, timeout, errp)`, where:

*mboxp* is a pointer to the intertask mailbox

*timeout* is the amount of time to wait for the message before timing out; 0 = use default timeout

*errp* is the error pointer

*rt\_pend* does the following:

- Calls *sc\_pend* to wait for the specified mailbox message. The timeout is set to the timeout specified in the call or to a default value.
- Calls *poll\_shutdown* to see if a system shutdown has been initiated.

The function returns the value returned from *sc\_pend* as result.

Called By:            *\_pend\_on\_frame\_interrupt*  
                      *\_rowcol\_rd*  
                      *bx\_task*  
                      *exchange\_flea\_data*  
                      *flea*  
                      *flea\_host\_if*  
                      *host\_if\_debug*  
                      *local\_terrain*  
                      *msg\_end*

Routines Called:    *poll\_shutdown*  
                      *sc\_pend*

Parameters:            *char*                                \*\* *mboxp*  
                      *long*                                *timeout*  
                      *int\**                                *errp*

Returns:                *result*

#### 2.13.41.2 *rt\_post*

The *rt\_post* function posts a message to a specified intertask mailbox by calling *sc\_post*.

The function call is *rt\_post(mboxp, msg, errp)*, where:

*mboxp* is a pointer to the mailbox

*msg* is a pointer to the message

*errp* is the error pointer

Called By:            *\_rowcol\_rd*  
                      *bx\_task*  
                      *local\_terrain*  
                      *scratch\_flea*

Routines Called:      `sc_post`

Parameters:            `char                    **mboxp`  
                 `char                    *msg`  
                 `int                     *errp`

Returns:                `none`

### 2.13.41.3    `rt_accept`

The `rt_accept` function clears any stray messages from a specified mailbox by calling `sc_accept`.

The function call is `rt_accept(mboxp, errp)`, where:

*mboxp* is a pointer to the mailbox  
*errp* is the error pointer

The function returns the value returned by `sc_accept`.

This function is not currently used.

Called By:              `none`

Routines Called:        `sc_accept`

Parameters:            `char                    **mboxp`  
                 `int                    *errp`

Returns:                `sc_accept(mboxp, errp)`

### 2.13.42      `rtd_init.c`

The functions in the `rtd_init.c` CSU are used to parse the command line entered by the user at startup, and to initialize the system. These functions are:

- `initialize_defaults`
- `initialize`

#### 2.13.42.1    `initialize_defaults`

The `initialize_defaults` function initializes various system variables to their default values. Currently, these variables are the CIG-SIM frame rate and the boolean that controls single-step mode.

The function call is **initialize\_defaults()**.

Called By:            initialize

Routines Called:     none

Parameters:           none

Returns:              none

### 2.13.42.2 initialize

The initialize function parses the command line entered by the user to start up the system. Command line arguments are used to select the correct network interface as well as to toggle certain types of processing on and off. initialize then calls other functions to initialize the system in the desired manner.

The arguments that can be entered on the initialize command line are listed below, along with the variable(s) set if that argument is specified.

Switch	Description	Variable(s)
A	AGPT mode - host via socket to simnet server	agpt_mode
a	Aerial terrain feedback shutoff (altitude above ground)	feedback_shutoff_G
b	Ballistics processing disabled	ball_disable_wanted
c	Constrain database to initial load	database_disable_wanted
d	Download texture maps to IPPTs	esifa_load_wanted
e	Ethernet mode	enet_mode_G
f	Flea mode	flea_mode_G
h	Help	none
m	MPV mode	mpv_mode_local
p	Set CIG-SIM packet exchange size	dr11in_pkt_size, dr11out_pkt_size
s	Set speed (frame rate in Hz)	frame_rate
S	STG slaved	slave_stg
t	Set local terrain chunk size and interval	lt_chunk_size, lt_interval
v	Verbose	verbose_G
x	Socket interface mode	local_socket_name
z	SCSI mode	scsi_mode_local



initialize does the following:

- Calls `initialize_defaults` to initialize various system parameters.
- Calls `clparse` to verify that the command line is valid.
- Calls `host_if_buffer_init` to reset the INBUF and OUTBUF buffers.
- Calls `host_init_packet_sizes` to initialize the CIG-SIM exchange packet sizes.
- Initializes the host interface based on the command line, as follows:
  - If Ethernet mode is requested ("e" argument), calls `init_enet_interface`.
  - If flea mode is requested, ("f" argument) calls `init_flea_interface`.
  - If MPV mode is requested ("m" argument), calls `init_mpv_interface`.
  - If SCSI mode is requested ("z" argument), calls `init_scsi_interface`.
  - If socket mode is requested ("x" argument), calls `init_socket_interface`.
  - If DR11 mode is requested (default), calls `init_dr11_interface`.
- Opens and parses the `subsys.cfg` file:
  - Calls `init_subsys_parser` to initialize the parser used to read the file.
  - Calls `fopen` to open the file.
  - Calls `set_token_file` and `parse_subsys_file` to read the file.
  - Calls `fclose` to close the file.
  - Calls the `get_*` functions to retrieve the specified file names, and outputs them to stdout.
- Calls `sys_control_init` to initialize the frame rate in the EVC board.
- For subsystem 0 (AAM1):
  - Calls `backend_setup` to configure the backend.
  - Calls `find_fn` to find the forcetask file ("force0").
  - Calls `mpvideo_boot` to start the MPV.
  - Calls `mpvideo_load` to load the 3-D and final lookup tables, the 2-D overlay file, and the 2-D task.
- If subsystem 1 (AAM2) exists:
  - Calls `backend_setup` to configure the backend.
  - Calls `find_fn` to find the forcetask file ("force1").
  - Calls `mpvideo_boot` to start the MPV.
  - Calls `mpvideo_load` to load the 3-D and final lookup tables, the 2-D overlay file, and the 2-D task.
- Calls `sys_control_init` to initialize the frame rate in the EVC board.
- Calls `sys_slave_sync` to synchronize the frame interval with an external master CIG, or `sys_master_sync` to set up the local CIG as the master.
- Calls `vpt_tree_init` to initialize the viewport configuration tree.

The function exits with 1 if an error was detected on the command line, or if `mpvideo_load` could not load a required file.

Called By:            `main` (in `rtt.c`)

Routines Called:    `backend_setup`  
                     `clparse`  
                     `exit`  
                     `fclose`  
                     `find_fn`  
                     `fopen`  
                     `get_3d_lut_name`  
                     `get_color_config_name`  
                     `get_data_2d_name`

```
get_database_name
get_ded_name
get_esifa_name
get_final_lut_name
host_if_buffer_init
host_init_packet_sizes
init_drll_interface
init_enet_interface
init_flea_interface
init_mpv_interface
init_scsi_interface
init_socket_interface
init_subsys_parser
initialize_defaults
mpvideo_boot
mpvideo_load
parse_subsys_file
printf
set_token_file
strcpy
strlen
sys_control_init
sys_master_sync
sys_slave_sync
vpt_tree_init
```

```
Parameters:      int      argc
                 char     *argv[]
```

**Returns:** none

### 2.13.43 simulation.c

The functions in the simulation.c CSU form the message handler for the real-time simulation control of the CIG hardware and communications with the Simulation Host. These functions are:

- **simulation**
- **process\_a\_msg**
- **syserr**

### 2.13.43.1 simulation

The simulation function calls other functions to initialize all simulation variables and process the messages received from the Simulation Host. This function is called when the Simulation Host sends a MSG\_CIG\_CTL message with the new state set to C\_SIMULATION or C\_MCC\_SIMUL.

The function call is **simulation(simstate)**, where *simstate* is the new state of the CIG system (C\_SIMULATION or C\_MCC\_SIMUL).

simulation does the following:

- Calls `init_simulation` to initialize various simulation variables and table pointers, notify the backend to prepare for a simulation, initiate database management and database feedback, and enable system interrupts.
- Calls `poll_shutdown` to see if a system shutdown has been requested.
- Makes sure the packet size has not been exceeded.
- Tests to see if an interrupt has occurred by looking for a `SIMULATION_MB` mailbox message. If so, flushes the frame by calling `sc_accept`. This step is skipped if the simulation is running under Flea.
- Calls `process_a_msg` to process each runtime message received from the Simulation Host.
- Calls `stop_watch` to read and stop the timer.

Called By: `db_mcc_setup`

Routines Called: `asm`  
`GLOB`  
`init_simulation`  
`poll_shutdown`  
`printf`  
`process_a_msg`  
`sc_accept`  
`stop_watch`  
`syserr`

Parameters: `INT_2` `simstate`

Returns: `none`

#### 2.13.43.2 `process_a_msg`

The `process_a_msg` function handles all messages received by the Simulation Host during a simulation. This function is called when the CIG is ready to start processing messages.

The function call is `process_a_msg(curr_sim_buf, pdbase, pdynl, ptanks, pmodels, peffects)`, where:

*curr\_sim\_buf* is the double buffer the real-time software is currently loading  
*pdbase* is a pointer to the primary database control block  
*pdynl* is a pointer to the structure containing the current state of all dynamic models  
*ptanks* is the starting address of the tanks table  
*pmodels* is the starting address of the dynamic models table  
*peffects* is the starting address of the dynamic effects table

For each message it receives, `process_a_msg` does the following:

- Calls `cigsimio_msg_in` to write the message to a buffer (if message recording or debug display is enabled).
- Processes the message according to its type (see table below).

In most cases, `process_a_msg` simply calls a Message Interface (`msg_*`) function to handle the message. For messages that are to be forwarded to Ballistics, `process_a_msg` pushes the message onto the Ballistics message queue or calls a `sim_bal_*` routine.

The following table summarizes the processing performed by `process_a_msg` in response to each valid message type it receives from the Simulation Host. The first column lists the messages in alphabetical order. The second column briefly describes the purpose of the message (in *italics*), then lists the major steps performed by `process_a_msg` to process that message.

Message from SIM Host	Processing by <code>process_a_msg</code>
MSG_IROTATION	<i>Updates a single rotation of an hprxyzs matrix.</i> Calls <code>msg_1rotation</code> .
MSG_3ROTATIONS	<i>Updates the rotation portion of an hprxyzs matrix.</i> Calls <code>msg_3rotations</code> .
MSG_ADD_TRAJ_TABLE	<i>Defines a new trajectory table.</i> Pushes MSG_B0_ADD_TRAJ_TABLE message onto Ballistics message queue.
MSG_AGL_SETUP	<i>Toggles AGL processing on and off.</i> Sets <code>agl_wanted</code> in global memory.
MSG_AMMO_DEFINE	<i>Define ammunition maps.</i> Loads the specified ammunition maps into memory.
MSG_CALIBRATION_IMAGE	<i>Displays offset image, color image, or BBN logo.</i> Calls <code>msg_calibration_image</code> .
MSG_CANCEL_ROUND	<i>Tells Ballistics to cancel a fired round.</i> Pushes MSG_B0_CANCEL_ROUND message onto Ballistics message queue.
MSG_CIG_CTL	<i>Causes a transition to another performance state.</i> Calls <code>msg_cig_ctl</code> .
MSG_CLOUD_STATE	<i>Positions cloud models at a specified altitude.</i> Calls <code>cloud_update</code> .
MSG_DELETE_TRAJ_TABLE	<i>Tells Ballistics to delete a downloaded trajectory table.</i> Pushes MSG_B0_DELETE_TRAJ_TABLE message onto Ballistics message queue.
MSG_DR11_PKT_SIZE	<i>Specifies exchange packet parameters.</i> Calls <code>msg_dr11_pkt_size</code> .
MSG_END	<i>Signals end of packet buffer.</i> Calls <code>pretend_veh</code> ; calls <code>msg_end</code> .
MSG_GUN_OVERLAY	<i>Changes gun/gunner overlays.</i> Calls <code>m1_gun_overlay</code> or <code>m2_gun_overlay</code> , based on type in message.
MSG_HPRXYZS_MATRIX	<i>Updates a configuration node's matrix.</i> Calls <code>msg_hprxyzs_matrix</code> .
MSG_LT_STATE	<i>Changes the local terrain frame interval and chunk size.</i> Calls <code>lt_state</code> .

MSG_OTHERVEH_STATE	<i>Describes the state of all dynamic vehicles in the terrain.</i> Calls msg_otherveh_state.
MSG_PASS_ON	<i>Provides data to be passed on to a specific subsystem in the CIG (e.g., 2-D overlay processor).</i> Calls msg_pass_on.
MSG_PPM_DISPLAY_MODE	<i>Changes PPM display modes.</i> Calls msg_ppm_display_mode.
MSG_PPM_DISPLAY_OFFSET	<i>Changes PPM display offsets.</i> Calls msg_ppm_display_offset.
MSG_PPM_PIXEL_LOCATION	<i>Sets a new PPM pixel location.</i> Calls msg_ppm_pixel_location.
MSG_PPM_PIXEL_STATE	<i>Turns PPM pixel processing on or off.</i> Calls msg_ppm_pixel_state.
MSG_PROCESS_CHORD	<i>Tells Ballistics to process a chord.</i> Pushes MSG_B0_PROCESS_CHORD message onto Ballistics message queue.
MSG_PROCESS_ROUND	<i>Tells Ballistics to process a fired round.</i> Pushes MSG_B0_PROCESS_ROUND message onto Ballistics message queue.
MSG_PROCESS_ROUND48	<i>Tells Ballistics to process a fired round.</i> Calls msg_process_round48.
MSG_REQUEST_LASER_RANGE	<i>Asks for pixel depth for i, j position on screen.</i> Calls msg_laser_request_range.
MSG_REQUEST_POINT_INFO	<i>Asks for terrain characteristics at a specific point in the viewing range.</i> Calls sim_bal_req_pt_info.
MSG_ROT2x1_MATRIX	<i>Updates a configuration node's matrix.</i> Calls msg_rot2x1_matrix.
MSG_ROUND_FIRED	<i>Tells Ballistics that a round has been fired.</i> Pushes MSG_B0_ROUND_FIRED message onto Ballistics message queue; if rtsw_timing flag is enabled, computes processing time for message and updates worst time if applicable.
MSG_RTS4x3_MATRIX	<i>Updates a configuration node's matrix.</i> Calls msg_rts4x3_matrix.
MSG_SCALE	<i>Updates the scale portion of an hprxyzs matrix.</i> Calls msg_scale.
MSG_SHOW_EFFECT	<i>Used to show the effect of an impact on terrain or a vehicle.</i> Calls msg_show_effect.
MSG_SIO_WRITE	<i>Writes data to a serial device.</i> Calls sio_write.
MSG_STATICVEH_REM	<i>Removes a static vehicle from the local terrain.</i> Calls msg_staticveh_rem.
MSG_STATICVEH_STATE	<i>Adds a static vehicle to the local terrain.</i> Calls msg_staticveh_state.
MSG_SUBSYS_MODE	<i>Changes subsystem-specific parameters (e.g., color table, fade value, calibration mode).</i> Calls msg_subsys_mode.

MSG_TF_INIT_HDR	<i>Controls the way the CIG builds a terrain feedback message.</i> Pushes MSG_B0_TF_INIT_HDR message onto Ballistics message queue; if "own" vehicle, sets flags and frequency variables.
MSG_TF_INIT_PT	<i>Specifies a point for which terrain feedback is desired.</i> Pushes MSG_B0_TF_INIT_PT message onto Ballistics message queue.
MSG_TF_STATE	<i>Disables/enables terrain feedback processing; changes frequency of own vehicle feedback.</i> Pushes MSG_B0_TF_STATE message onto Ballistics message queue; if message is for "own" vehicle, sets flags and frequency accordingly.
MSG_TF_VEHICLE_POS	<i>Provides new position of vehicle for which terrain feedback data is being collected.</i> Pushes MSG_B0_TF_VEHICLE_POS message onto Ballistics message queue.
MSG_TRAJ_CHORD	<i>Used for chords that represent trajectories.</i> Calls sim_bal_traj_chord; if rtsw_timing flag is enabled, computes processing time for message and updates worst time if applicable.
MSG_TRAJ_ENTRY	<i>Downloads an entry to a trajectory table.</i> Pushes MSG_B0_TRAJ_ENTRY message onto Ballistics message queue.
MSG_TRANSLATION	<i>Updates the translation portion of an hprxyzs matrix.</i> Calls msg_translation.
MSG_VIEW_FLAGS	<i>Updates system view flags (e.g., on/off, FLIR, DTV) and/or branch values.</i> Calls msg_view_flags.
MSG_VIEW_MAGNIFICATION	<i>Changes viewport's field of-view and/or level of detail.</i> Calls msg_view_magnification.
MSG_VIEWPORT_UPDATE	<i>Modifies viewport parameters.</i> Calls msg_viewport_update.

Called By: simulation

Routines Called:

- asm
- cigsimio\_msg\_in
- cloud\_update
- GLOB
- lt\_state
- m1\_gun\_overlay
- m2\_gun\_overlay
- msg\_1rotation
- msg\_3rotations
- msg\_calibration\_image
- msg\_cig\_ctl
- msg\_dr11\_pkt\_size
- msg\_end
- msg\_hprxyzs\_matrix

```

msg_laser_request_range
msg_otherveh_state
msg_pass_on
msg_ppm_display_mode
msg_ppm_display_offset
msg_ppm_pixel_location
msg_ppm_pixel_state
msg_process_round48
msg_rot2x1_matrix
msg_rts4x3_matrix
msg_scale
msg_show_effect
msg_staticveh_rem
msg_staticveh_state
msg_subsys_mode
msg_translation
msg_view_flags
msg_view_magnification
msg_viewport_update
mx_push
pretend_veh
printf          (in debug mode only)
read_watch
sim_bal_req_pt_info
sim_bal_traj_chord
sio_write
syserr

```

Parameters:	INT_2	*curr_sim_buf
	DB_INFO	*pdbname
	INT_4	**pdynl
	TANK	**ptanks
	OMODEL	**pmodels
	SHOW_EFF	**peffects

Returns: none

### 2.13.43.3 syserr

The `syserr` function is used to generate a `MSG_SYS_ERROR` message for return to the Simulation Host. This function is called if an error is detected in a SIM-to-CIG message packet (e.g., packet length overrun or illegal message type), or a function requested by the Simulation Host could not be performed (e.g., configuration tree initialization error or laser range request failure).

The function call is `syserr(error, state)`, where:

*error* is the error that was detected  
*state* is the current state of the CIG

The MSG\_SYS\_ERROR message is included in every CIG-to-SIM message packet, even if no error has been detected. The syserr function is called only if an error occurs.

Called By:	_pend_on_frame_interrupt cig_config exchange_enet_data exchange_enet_data_sim lt_state m1_gun_overlay m2_gun_overlay mpvideo_response msg_cig_ctl msg_laser_request_range msg_pass_on msg_subsys_mode msg_viewport_update process_a_msg simulation	
Routines Called:	none	
Parameters:	INT_2 INT_2	error state
Returns:	none	

#### 2.13.44 staticveh\_remove.c

The staticveh\_remove function deletes a static vehicle from the simulation environment. This function is called (once per backend) whenever the Simulation Host sends a MSG\_STATICVEH\_REM message.

The function call is **staticveh\_remove(msgp, pdbase, vme\_offset, backend)**, where:

*msgp* is a pointer to the MSG\_STATICVEH\_REM message  
*pdbase* is a pointer to the primary database control block  
*vme\_offset* is the VME address offset to active area memory  
*backend* is the backend id

staticveh\_remove does the following:

- Uses FIND\_LM to determine the load module that contains the vehicle, based on its coordinates.
- Determines which list to remove the model from (pre-process, post-process, or standard process).
- If the backend is 0, calls sim\_bal\_static\_rem to notify Ballistics to remove the vehicle from its tables.



- Deletes the vehicle from the applicable table, and decrements all associated counters.
- Removes any special effects associated with the vehicle.

Called By: msg\_staticveh\_rem

Routines Called: FIND\_LM  
panic  
sim\_bal\_static\_rem

Parameters:	MSG_STATICVEH_REM	*msgp
	DB_INFO	*pdbname
	INT_4	vme_offset
	INT_4	backend

Returns: none

#### 2.13.45 staticveh\_state.c

The staticveh\_state function adds a static vehicle to the simulation environment. This function is called (once per backend) whenever the Simulation Host sends a MSG\_STATICVEH\_STATE message.

The function call is **staticveh\_state(msgp, dbname, vme\_offset, backend)**, where:

*msgp* is a pointer to the MSG\_STATICVEH\_STATE message  
*dbname* is a pointer to the primary database control block  
*vme\_offset* is the VME address offset to active area memory  
*backend* is the backend id

staticveh\_state does the following:

- Determines whether the vehicle is a tank; sets the veh\_is\_tank flag appropriately.
- Adds the vehicle to the appropriate table (tank or model).
- Increments the applicable model counter.
- Saves the model's matrix and center.
- Uses FIND\_LM to determine the load module that contains the vehicle, based on its coordinates.
- Adds the vehicle to the appropriate load module table.
- Adds the vehicle to the appropriate model list (pre-process, post-process, or standard process).
- If the backend is 0, calls sim\_bal\_static\_add to notify Ballistics to add the vehicle to its tables.

The function returns 0 if successful. It returns E\_LIMIT if there is no room for the new tank or model.

Called By: msg\_staticveh\_state

Routines Called:	FIND_LM sim_bal_static_add	
Parameters:	MSG_STATICVEH_STATE DB_INFO INT_4 INT_4	*msgp *pdbname vme_offset backend
Returns:	0 E_LIMIT	

### 2.13.46 sys\_control.c

The functions in the sys\_control.c CSU interface with the EVC (Ethernet VME Controller) board. The purpose of this board is to allow frame interrupts onto the VME bus.

These functions are:

- sys\_control\_init
- sys\_frame\_rate
- set\_leds
- sys\_slave\_sync
- sys\_master\_sync
- get\_dtp\_bank
- read\_evc\_control
- write\_evc\_control
- write\_evc\_frame

The typedef for the EVC structure is also defined in this CSU. The EVC structure contains the following:

- Interrupt control register (icr).
- Interrupt vector (ivr).
- LED display value.
- System control register. This is a control word that describes the EVC mode, CIG mode (master or slave), the current EVC bank (the double buffer in use by the hardware this frame), and the frame rate.
- Single\_shot system frame register, used for <<TBD>>.

#### 2.13.46.1 sys\_control\_init

The sys\_control\_init function initializes frame interrupts. This function is called when the system is initialized. It is also called through Gossip if the user selects the i ("initialize EVC") option from the GT Hardware menu, reached through the Gossip System menu.

The function call is sys\_control\_init(prate, ptime), where:

*prate* is a pointer to the CIG's frame rate; this can be entered by the user on the command line at startup; if no speed is specified, the initialize function sets a default value

*ptime* is a pointer to the frame time; this is calculated by `sys_frame_rate`

`sys_control_init` does the following:

- Sets `use_evc` to FALSE.
- Calls `sysrup_off` to disable system interrupts.
- Calls `bus_error` to look for the EVC board; if found, sets `use_evc` to TRUE.
- Sets the EVC's interrupt vector.
- Initializes the EVC's control register to normal mode.
- Calls `sys_master_sync` to set the EVC to operate as the master frame interrupt.
- Calls `sys_frame_rate` to set the EVC's frame rate.
- Sets the EVC's interrupt control register.

The function exits with a 1 if it cannot find the EVC board.

Called By: `gos_system  
initialize`

Routines Called: `bus_error  
exit  
printf  
sys_frame_rate  
sys_master_sync  
sysrup_off`

Parameters: `INT_4                   *prate  
INT_4                   *ptime`

Returns: `none`

#### 2.13.46.2 `sys_frame_rate`

The `sys_frame_rate` function sets the desired frame rate (10, 15, 30, or 60 Hz) in the EVC's system control register. The frame rate set is based on the speed entered by the operator on the command line (using the "s" argument) at startup. If no speed was specified, the default selected by the initialize function is used.

The function call is `sys_frame_rate(prate, ptime)`, where:

*prate* is a pointer to the frame rate

*ptime* is a pointer to the frame time; this is calculated by `sys_frame_rate`

If an invalid frame rate is specified in the call, `sys_frame_rate` assigns a default of 30. `sys_frame_rate` also calculates the frame time based on the frame rate.

Called By:	sys_control_init	
Routines Called:	printf	
Parameters:	INT_4 INT_4	*prate *ptime
Returns:	none	

### 2.13.46.3 set\_leds

The set\_leds function displays a value on the EVC's LED display. This function is called at the end of every frame during a simulation. It is also invoked through Gossip if the user selects the 1 ("write into leds") option from the GT Hardware menu, reached through the Gossip System menu.

The function call is set\_leds(value), where value is the value to be displayed on the LED.

The EVC LED display can show any of the following values:

- Model count.
- Frame count.
- Local terrain frame count.
- Overload indicator (appears only if a system overload is detected)

The display\_lights[] array in global memory contains flags that enable/disable each of these counters to appear on the LED. The array is checked at the end of each frame to see which counts are to be passed to set\_leds to display. The flags in display\_lights[] can be changed using the 1 ("set display lights flags") option on the Gossip System menu.

Called By:	_display_lights gos_system	
Routines Called:	none	
Parameters:	INT_4	value
Returns:	none	

### 2.13.46.4 sys\_slave\_sync

The sys\_slave\_sync function synchronizes the CIG's frame interval with an external master CIG. This function is called at startup if the local CIG is operating in a slave mode. Slave mode is specified by entering the "S" argument on the startup command line.

The function call is **sys\_slave\_sync()**.

Called By: initialize

Routines Called: none

Parameters: none

Returns: none

#### 2.13.46.5 sys\_master\_sync

The **sys\_master\_sync** function sets the local EVC to operate as the master frame interrupt. This function is called at startup if the CIG is operating in master mode (the default).

The function call is **sys\_master\_sync()**.

Called By: initialize  
sys\_control\_init

Routines Called: none

Parameters: none

Returns: none

#### 2.13.46.6 get\_dtp\_bank

The **get\_dtp\_bank** function returns the number of the double buffer currently being used by the Data Traversal Processor (DTP). The real-time software then sets the current simulation buffer to the opposite buffer. (The real-time software and the DTP alternate double buffers each frame. This provides the most recent data to the hardware while the software updates the other buffer.) This function is called at the beginning of a new simulation, and at the end of every frame during a simulation.

The function call is **get\_dtp\_bank()**.

The function obtains the bank number from the EVC's system control register. It returns EOF if the **use\_evc** flag is false, indicating that the EVC board has not been initialized.

Called By: \_reset\_model\_pointers  
init\_simulation

Routines Called:     printf

Parameters:           none

Returns:             EOF  
                      pevc->control\_r & EVC\_BANK

#### 2.13.46.7   read\_evc\_control

The `read_evc_control` function returns the contents of the EVC's system control register (`pevc->control_r`). This function is called through Gossip if the user selects the `d` ("display EVC register") option from the GT Hardware menu, reached through the Gossip System menu.

The function call is `read_evc_control()`.

The function returns the data in the control register as *value*. The *value* is set to EOF if the `use_evc` flag is false, indicating that the EVC board is not initialized.

Called By:            gos\_system

Routines Called:     printf

Parameters:           none

Returns:             value

#### 2.13.46.8   write\_evc\_control

The `write_evc_control` function writes a specified value to the EVC's system control register (`pevc->control_w`). This function is called through Gossip if the user selects the `c` ("write into EVC frame rate control word") option from the GT Hardware menu, reached through the Gossip System menu.

The function call is `write_evc_control(value)`, where *value* is the value to be written.

Called By:            gos\_system

Routines Called:     printf

Parameters:           INT\_4                             value

Returns: none

### 2.13.46.9 write\_evc\_frame

The `write_evc_frame` function writes a specified value to the EVC's system frame register (`pevc->single_shot`). This function is called through Gossip if the user selects the `r` ("write into EVC single shot word") from the GT Hardware menu, reached through the Gossip System menu.

The function call is `write_evc_frame(value)`, where *value* is the value to be written.

Called By: `gos_system`

Routines Called: `printf`

Parameters: `INT_4` `value`

Returns: none

### 2.13.47 upstart.c

The `upstart.c` CSU contains the driver for the real-time software. The functions in this CSU are:

- `upstart`
- `upstart_cleanup`

#### 2.13.47.1 upstart

The `upstart` function is the driver for the real-time applications software. It establishes communication with the Simulation Host, reads a message, then calls the appropriate function based on the system state requested in the message.

The function call is `upstart()`. `upstart` does the following:

- Calls `start_watch` to start the system timer.
- Initializes the `local_terrain_wanted` variable to `TRUE`.
- Calls `poll_shutdown` to see if a system shutdown has been requested.
- Processes each message received from the Simulation Host:
  - Increments the packet buffer size; checks for packet buffer overflows.
  - Calls `cigsimio_msg_in` to write the message to a buffer (if message recording or debug display is enabled).
  - Processes the message according to its type (see table below).

The following table summarizes the processing performed by upstart in response to each valid message type it receives from the Simulation Host. The first column lists the messages in alphabetical order. The second column briefly describes the purpose of the message (in *italics*), then lists the major steps upstart performs to process the message.

Message from SIM Host	Processing by upstart
MSG_AMMO_DEFINE	<i>Defines ammunition maps.</i> Loads the specified ammo_map tables into memory.
MSG_CIG_CTL C_DB_SETUP C_FILE_XFER C_MCC_SETUP C_NULL C_STOP C_TEST_MODE	<i>Causes a transition to another performance state.</i> Calls db_mcc_setup with state set to C_DB_SETUP. Calls file_control with state set to C_FILE_XFER. Calls db_mcc_setup with state set to C_MCC_SETUP. No action. No action. Calls hw_test with state set to C_TEST_MODE.
MSG_DR11_PKT_SIZE	<i>Specifies exchange packet parameters.</i> Outputs data from message to stdout; sets CIG and SIM exchange packet sizes; validates local terrain chunk size and message interval, and sets if valid; sets CIG hardware type.
MSG_END	<i>Signals end of packet buffer.</i> Calls cigsimio_msg_out to write outgoing message packet to buffer if recording is enabled; calls start_watch; calls appropriate exchange_data routine (through *exchange_data) to exchange packets.
MSG_PPM_DISPLAY_MODE	<i>Changes PPM display modes.</i> Calls msg_ppm_display_mode.
MSG_PPM_DISPLAY_OFFSET	<i>Changes PPM display offsets.</i> Calls msg_ppm_display_offset.
MSG_PPM_PIXEL_LOCATION	<i>Sets a new PPM pixel location.</i> Calls msg_ppm_pixel_location.
MSG_PPM_PIXEL_STATE	<i>Turns PPM pixel processing on or off.</i> Calls msg_ppm_pixel_state.

Called By: none (task created and started at initialization time)

Routines Called:

- \*exchange\_data
- cigsimio\_msg\_in
- cigsimio\_msg\_out
- db\_mcc\_setup
- file\_control
- GLOB
- hw\_test
- msg\_ppm\_display\_mode
- msg\_ppm\_display\_offset
- msg\_ppm\_pixel\_location
- msg\_ppm\_pixel\_state
- poll\_shutdown
- printf



start\_watch  
SYSERR

Parameters: none

Returns: none

### 2.13.47.2 upstart\_cleanup

The upstart\_cleanup function deallocates the resources owned by the upstart task. This function is called when a system shutdown is requested by the Gossip user. The function is called via the \*task\_cleanup function pointer, which points to the cleanup routine's name in the task table.

The function call is **upstart\_cleanup()**.

The function returns 1 if successful, or 0 if an error occurred.

*Note: This function is not yet implemented. At the current time, it simply returns a 1 if called.*

Called By: poll\_shutdown (through \*task\_cleanup)

Routines Called: none

Parameters: none

Returns: 0  
1

## 2.14 Serial Device Input/Output (/cig/libsrc/libcio)

The Serial Device Input/Output CSC lets the Simulation Host write to a serial I/O device during a simulation. Messages are used to open, write to, and close the device. This feature can be used to record simulation-related data to a serial printer, or to drive a serial device that performs sort of processing during the simulation.

A maximum of four serial devices can be open during one simulation. For each device, the Simulation Host specifies the following:

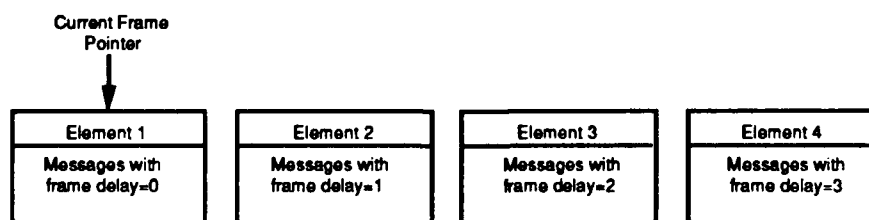
- The device name. This is used to establish an output channel to the device.
- A unique identifier (0-3).
- The maximum number of frames to delay before actually sending data to the device. This delay allows synchronization of data written to the device with outputs from other parts of the system.
- The maximum number of characters to be written to the device during any given frame.

Each write message specifies the following:

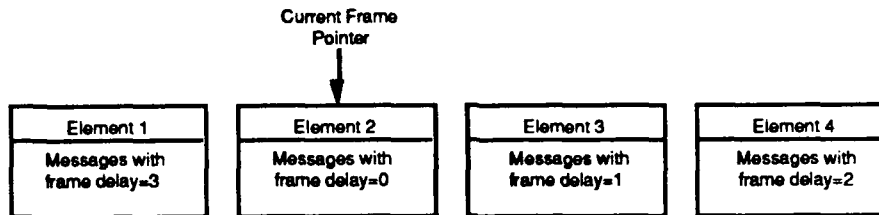
- The data to be sent.
- The identifier of the device to which the data is to be written.
- The number of frames to delay before writing the data. This delay must be less than or equal to the maximum delay specified for the device.

All data to be written to a device is buffered in a local queue until the specified frame delay expires. The queue is implemented as an array, with the array size set equal to the maximum frame delay plus 1. Each element in a device's array contains the data to be written to that device during a particular frame.

For example, a device with a maximum frame delay of 3 has an output queue array containing four elements. When the Simulation Host sends a write message, a Serial Device I/O function determines (based on the message's frame delay) which element the data is to be stored in. If it reaches the end of the array, it wraps back around to the first element.



At the end of each frame, a tick function accesses the current frame's element in each device's array, and writes all of the stored data to that device. The data in the element may come from multiple write messages. The tick function then moves the current frame pointer to the next element, for use during the next frame.



As shown in Figure 2-18, the Serial Device Input/Output CSC consists of only one CSU, `sio.c`. This CSU is described in this section.

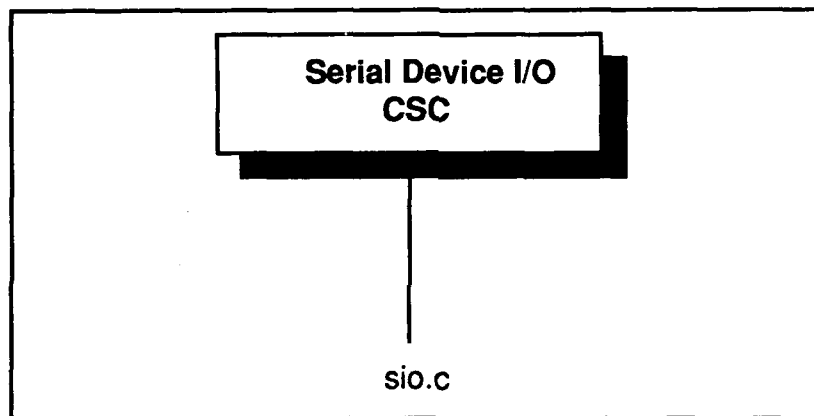


Figure 2-18. Serial Device Input/Output CSU

### 2.14.1 `sio.c`

The functions in the `sio.c` CSU handle all communication with the serial devices. These functions are:

- `sio_init`
- `sio_write`
- `sio_close`
- `sio_tick`

#### 2.14.1.1 `sio_init`

The `sio_init` function opens a channel to a serial device. This function is called when the Simulation Host sends a `MSG_SIO_INIT` message during the "prepare for simulation" state. The message specifies the device name, an identifier to associate with the device, the maximum frame delay, and the maximum number of characters to be written during any one frame.

The function call is **sio\_init(msgp)**, where *msgp* is a pointer to the MSG\_SIO\_INIT message.

**sio\_init** does the following:

- Verifies that the identifier is valid.
- Makes sure the specified identifier is not already in use by another open device.
- Opens the device in write-only mode.
- Allocates memory for an output queue for the device; the number of array elements is set equal to one more than the maximum frame delay specified in the message.
- Allocates memory for each queue element (one per frame); the size of each element is set equal to the maximum number of characters per frame specified in the message.

The function returns 0 if successful. If an error occurs, the function returns *err* set to one of the following values:

14 (E_SIODEVID)	Invalid identifier.
15 (E_SIOIDUSED)	Identifier already assigned.
17 (E_SIOOPEN)	Error opening device.
18 (E_SIOMEM)	Insufficient memory for output queue.

Called By: db\_mcc\_setup

Routines Called: calloc  
free  
ifx\_open  
malloc

Parameters: MSG\_SIO\_INIT \*msgp

Returns: 0  
err

#### 2.14.1.2 sio\_write

The **sio\_write** function places the data sent by the Simulation Host for a serial device into the appropriate frame element of the device's output queue. This function is called when the Simulation Host sends a MSG\_SIO\_WRITE message. The message specifies the data, the data length in bytes, and the frame delay.

Data from multiple write messages to the same device to be output during the same frame is concatenated together.

The function call is **sio\_write(msgp)**, where *msgp* is a pointer to the MSG\_SIO\_WRITE message.

**sio\_write** does the following:

- Makes sure the specified identifier is valid.
- Makes sure the specified identifier is in use by an open device.
- Makes sure the specified frame delay does not exceed the device's maximum frame delay.
- Makes sure there is room in the specified frame element for the additional data (i.e., that the maximum length per frame specified by the Simulation Host will not be exceeded).
- Copies the data to the appropriate frame element in the device's output queue. If the message's frame delay is 1, for example, the data is written to the next element after the current one. When the end of the array is reached, `sio_write` wraps back around to the beginning.
- Increments the character count for the selected queue element by the size of the new data.

The function returns 0 if successful. If an error occurs, the function returns *err* set to one of the following values:

14 (E_SIODEVID)	Invalid identifier.
16 (E_SIOIDNOTUSED)	No currently open device has this identifier.
20 (E_SIOOVFLO)	Output queue overflow.
21 (E_SIOFRAME)	Invalid frame delay.

Called By: `process_a_msg`

Routines Called: `bcopy`

Parameters: `MSG_SIO_WRITE` `*msgp`

Returns: `0`  
`err`

### 2.14.1.3 `sio_close`

The `sio_close` function closes a channel to a specified serial device, and frees the memory allocated to its output queue. This function is called if the Simulation Host sends a `MSG_SIO_CLOSE` message, which is valid only during the "prepare for simulation" state.

The function call is `sio_close(msgp)`, where *msgp* is a pointer to the `MSG_SIO_CLOSE` message.

The function does the following:

- Makes sure the identifier is valid.
- Makes sure the device is currently open (i.e., that the identifier is flagged in-use).
- Frees the memory allocated to the device's output queue.
- Closes the channel to the device.
- Marks the identifier as not in-use.

The function returns 0 if successful. If an error occurs, the function returns *err* set to one of the following values:

14 (E_SIODEVID)	Invalid identifier.
16 (E_SIOIDNOTUSED)	No currently open device has this identifier.
22 (E_SIOCLOSE)	Error closing device.

Called By: db\_mcc\_setup

Routines Called: free  
ifx\_close

Parameters: MSG\_SIO\_CLOSE \*msgp

Returns: 0  
err

#### 2.14.1.4 sio\_tick

The `sio_tick` function writes the queued data to each open device. This function is called at the end of every frame while the CIG is in the simulation state. All data queued for a given device that has reached its specified frame delay is written to the device. Note that this data may be from multiple messages.

The function call is `sio_tick()`. For each identifier that is flagged as in-use, `sio_tick` does the following:

- Determines whether the current frame element in the device's queue contains data to write.
- Writes all data in the current frame element to the device.
- Resets the character count in the current frame element to 0.
- Increments the queue index (the pointer to the current queue element in the array) for use the next frame.

The function returns 0 if successful. If a write error occurs, the function returns *err* set to 19 (E\_SIOWRITE).

Called By: msg\_end

Routines Called: ifx\_write

Parameters: none

Returns: 0  
err

## 2.15 Token Processing (/cig/libsrc/libtoken)

The Token Processing CSC functions are used to parse and process the data in the subsystem configuration (subsys.cfg) file. This file, which is read at initialization time, identifies the data files to be used for the simulation exercise.

The default subsystem configuration file is subsys.cfg. The Simulation Host can load a different configuration file by sending a MSG\_FILE\_DESCR message with *db\_req* set to DB\_SUBSYS\_0 or DB\_SUBSYS\_1. (The 0 or 1 identifies the backend.) Similarly, the Simulation Host can load an individual database file by sending a MSG\_FILE\_DESCR message with *db\_req* set to the appropriate file type.

Each subsystem (backend) has its own set of database files. An array indexed by subsystem id is maintained for each file name. Separate functions are provided to set or return each file name.

The files identified in subsys.cfg, and the MSG\_FILE\_DESCR messages that can be used to specify them, are the following:

File Name Array	File Contents	MSG_FILE_DESCR db_req setting
database_name[]	Terrain database	DB_SETUP_0 DB_SETUP_1
ded_name[]	Dynamic Elements Database (DED)	DB_DED_SETUP_0 DB_DED_SETUP_1
final_lut_name[]	Final (2-D) lookup table	DB_FINAL_LUT_0 DB_FINAL_LUT_1
3d_lut_name[]	3-D lookup table	DB_3D_LUT_0 DB_3D_LUT_1
data_2d_name[]	2-D overlay configuration data	DB_2D_DATA_0 DB_2D_DATA_1
esifa_name[]	ESIFA textures	DB_ESIFA_LOAD_0 DB_ESIFA_LOAD_1
color_config_name[]	Color configuration data	DB_COLOR_CFG_0 DB_COLOR_CFG_1

Figure 2-19 identifies the CSUs in the Token Processing CSC.

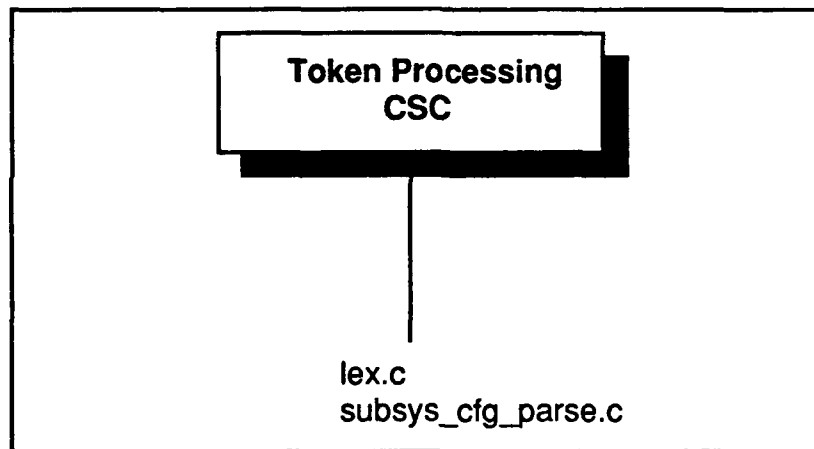


Figure 2-19. Token Processing CSUs

### 2.15.1 lex.c

The functions in the lex.c CSU are lexical utilities used to parse the subsystem configuration file. These functions are:

- set\_token\_file
- next\_char
- scan\_number
- next\_token
- next\_line
- swallow\_token
- get\_next\_token
- get\_current\_token
- get\_number\_value
- get\_string\_value

The subsystem configuration file is composed of tokens followed by numeric or string values. All recognized tokens are defined in the lex.h and keywords.h include files.

#### 2.15.1.1 set\_token\_file

The set\_token\_file function gets the name of the subsystem configuration file and sets it for the other Token Processing functions to use. It also initializes the work buffer and sets the buffer size. This function is called at start-up if a subsys.cfg file is found. It is also called if the Simulation Host sends a MSG\_FILE\_DESCR message with *db\_req* set to DB\_SUBSYS\_0 or DB\_SUBSYS\_1.

The function call is **set\_token\_file(file\_ptr)**, where *file\_ptr* is a pointer to the subsystem configuration file.

Called By: db\_mcc\_setup  
initialize



Routines Called: none

Parameters: FILE \*file\_ptr

Returns: none

#### 2.15.1.2 next\_char

The `next_char` function fetches the next character from the file and sets a pointer (`input_char`) to it. If the buffer is empty, `next_char` reads in a new buffer of data.

The function call is `next_char()`.

If `next_char` finds the end of the file, it sets the input character to -1 and sets the end-of-file flag (`at_eof`) to TRUE.

Called By: next\_token  
scan\_number

Routines Called: fread

Parameters: none

Returns: none

#### 2.15.1.3 scan\_number

The `scan_number` function scans over a number and converts it to binary. This function is called whenever the input character is a digit from 0 through 9.

The function call is `scan_number()`. The function converts the digit, then calls `next_char` to get the next digit. It continues to convert each digit until it receives a character outside the range 0 through 9.

Called By: next\_token

Routines Called: next\_char

Parameters: none

Returns: none

#### 2.15.1.4 next\_token

The `next_token` function examines the current input character (or a string of consecutive characters) and sets the variable *token* accordingly.

The function call is `next_token()`. The function does the following:

- Checks to see if the input character is a space; calls `next_char` to get another character if it is.
- Examines the input character.
- If the input character is not a letter, sets *token* to the appropriate recognized token.
  - In some cases, calls `next_char` to determine to correct token (e.g., checks for "=" after ">", and checks for a number after a minus sign).
  - If the character is between 0 and 9, calls `scan_number` to convert it to binary.
  - If the character is -1 (`next_char`'s signal for end-of-file), sets *token* to `tokenEOF`.
- If the input character is a letter:
  - Puts the letter into an array.
  - Calls `next_char` to get the next character and adds it to the array.
  - Compares the array with the list of valid words.
  - Keeps adding letters up to the maximum length allowed, looking for a match.
  - If a match is found, sets *token* to the matching word.
  - If no match is found, sets *token* to `tokenString`.

Called By:            `get_next_token`  
                      `next_line`  
                      `parse_subsys_file`  
                      `swallow_token`

Routines Called:    `next_char`  
                      `scan_number`  
                      `strcmp`

Parameters:                none

Returns:                    none

#### 2.15.1.5 next\_line

The `next_line` function skips to the first token on the next line of the file. It calls `next_token` until it finds a `tokenEOL` token, then calls `next_token` once more so *token* is set to the first token on the following line.

The function call is `next_line()`.

Called By: none

Routines Called: next\_token

Parameters: none

Returns: none

#### 2.15.1.6 swallow\_token

The `swallow_token` function checks that a particular token is present, then skips it by calling `next_token`. This function is used to verify that the correct starting and ending tokens are present in the file.

The function call is `swallow_token(tok)`, where *tok* is the token expected next in the file.

`swallow_token` returns 0 if the specified token was found. If a different token was found, the function outputs an error and returns 1.

Called By: parse\_subsys\_file

Routines Called: get\_string\_value  
next\_token  
printf

Parameters: int tok

Returns: 0  
1

#### 2.15.1.7 get\_next\_token

The `get_next_token` function calls `next_token` to read the next token from the file. It then returns the new value of *token*.

The function call is `get_next_token()`.

Called By: parse\_subsys\_file

Routines Called: next\_token

Parameters: none

Returns: token

#### 2.15.1.8 get\_current\_token

The `get_current_token` function returns the current value of *token*.

The function call is `get_current_token()`.

Called By: parse\_subsys\_file

Routines Called: none

Parameters: none

Returns: token

#### 2.15.1.9 get\_number\_value

The `get_number_value` function returns the current value of *token\_num\_value*. This value is set by `scan_number` and `next_token`. This function is used to get the subsystem (backend) id specified in the configuration file.

The function call is `get_number_value()`.

Called By: parse\_subsys\_file

Routines Called: none

Parameters: none

Returns: token\_num\_value

#### 2.15.1.10 get\_string\_value

The `get_string_value` function returns the current value of *token\_string\_value*. This value is set by `next_token` when parsing the command lines in the subsystem configuration file. This function is used to determine which file name is being set, as well as to get the file name.

The function call is **get\_string\_value()**.

Called By:	parse_subsys_file swallow_token
Routines Called:	none
Parameters:	none
Returns:	token_string_value

### 2.15.2      **subsys\_cfg\_parse.c**

The functions in the `subsys_cfg_parse.c` CSU are used to read the information in the `subsys.cfg` configuration file. Routines are provided to get and set each file name.

The functions in this CSU are:

- `init_subsys_parser`
- `parse_subsys_file`
- `get_database_name`
- `get_ded_name`
- `get_final_lut_name`
- `get_3d_lut_name`
- `get_data_2d_name`
- `get_esifa_name`
- `get_color_config_name`
- `set_database_name`
- `set_ded_name`
- `set_final_lut_name`
- `set_3d_lut_name`
- `set_data_2d_name`
- `set_esifa_name`
- `set_color_config_name`

#### 2.15.2.1      **init\_subsys\_parser**

The `init_subsys_parser` function initializes the names of all database configuration files for all backends to blanks. This function is called at startup, before the `subsys.cfg` file is opened and processed. It is also called if the Simulation Host sends a `MSG_FILE_DESCR` message with `db_req` set to `DB_SUBSYS_0` or `DB_SUBSYS_1` (before opening and processing the file specified in the message).

The function call is **init\_subsys\_parser()**.

Called By:	db_mcc_setup initialize
------------	----------------------------

Routines Called:      strcpy

Parameters:            none

Returns:                none

#### 2.15.2.2    parse\_subsys\_file

The parse\_subsys\_file function reads the subsystem configuration file and sets the database file name variables with the file names specified in the file. This function is called at system initialization time to read and process the subsys.cfg file. It is also called if the Simulation Host sends a MSG\_FILE\_DESCR message with *db\_req* set to DB\_SUBSYS\_0 or DB\_SUBSYS\_1.

The function call is parse\_subsys\_file(). The function does the following:

- Verifies that the file begins with the token\_start and token\_subsystem tokens.
- Calls get\_number\_value to determine the subsystem (backend) id.
- Validates the subsystem id.
- Calls next\_token to read each token in the file.
  - Makes sure the token is followed by a file name.
  - Copies the file name to the subsystem's element in the appropriate file name array.
- Stops processing when the token\_end token is found.

The function returns ERROR if the file format is invalid, an unknown token is found, a token is not followed by a file name, or the subsystem id is invalid.

Called By:              db\_mcc\_setup  
                          initialize

Routines Called:        get\_current\_token  
                          get\_next\_token  
                          get\_number\_value  
                          get\_string\_value  
                          next\_token  
                          printf  
                          strcpy  
                          swallow\_token

Parameters:            none

Returns:                1 (ERROR)

### 2.15.2.3 get\_database\_name

The `get_database_name` function returns the name of the terrain database file for a specified subsystem. Terrain database file names are stored in the `database_name[]` array, indexed by subsystem id.

The function call is `get_database_name(subsys)`, where *subsys* is the backend id.

Called By:	config_database db_mcc_setup initialize
Routines Called:	none
Parameters:	int subsys
Returns:	database_name[subsys]

### 2.15.2.4 get\_ded\_name

The `get_ded_name` function returns the name of the Dynamic Elements Database file for a specified subsystem. DED file names are stored in the `ded_name[]` array, indexed by subsystem id.

The function call is `get_ded_name(subsys)`, where *subsys* is the backend id.

Called By:	config_database db_mcc_setup initialize
Routines Called:	none
Parameters:	int subsys
Returns:	ded_name[subsys]

### 2.15.2.5 get\_final\_lut\_name

The `get_final_lut_name` function returns the name of the final (2-D) lookup table file for a specified subsystem. Final lookup table names are stored in the `final_lut_name[]` array, indexed by subsystem id.

The function call is **get\_final\_lut\_name(subsys)**, where *subsys* is the backend id.

Called By: db\_mcc\_setup  
initialize

Routines Called: none

Parameters: int subsys

Returns: final\_lut\_name[subsys]

#### 2.15.2.6 get\_3d\_lut\_name

The **get\_3d\_lut\_name** function returns the name of the 3-D color lookup table file for a specified subsystem. 3-D lookup table file names are stored in the **lut\_3d\_name[]** array, indexed by subsystem id.

The function call is **get\_3d\_lut\_name(subsys)**, where *subsys* is the backend id.

Called By: db\_mcc\_setup  
initialize

Routines Called: none

Parameters: int subsys

Returns: lut\_3d\_name[subsys]

#### 2.15.2.7 get\_data\_2d\_name

The **get\_data\_2d\_name** function returns the name of the 2-D overlay data file for a specified subsystem. 2-D overlay database file names are stored in the **data\_2d\_name[]** array, indexed by subsystem id.

The function call is **get\_data\_2d\_name(subsys)**, where *subsys* is the backend id.

Called By: db\_mcc\_setup  
initialize

Routines Called: none



**Parameters:**           int                                 subsys

**Returns:** `data_2d_name[subsys]`

### 2.15.2.8 get\_esifa\_name

The `get_esifa_name` function returns the name of the ESIFA textures file for a specified subsystem. Textures file names are stored in the `esifa_name[]` array, indexed by subsystem id.

The function call is `get_esifa_name(subsys)`, where *subsys* is the backend id.

**Called By:** db\_mcc\_setup  
initialize

Routines Called: none

**Parameters:**           int                                 subsys

**Returns:** `esifa_name[subsys]`

### 2.15.2.9 `get_color_config_name`

The `get_color_config_name` function returns the name of the color configuration file for a specified subsystem. Color configuration file names are stored in the `color_config_name[]` array, indexed by subsystem id.

The function call is `get_color_config_name(subsys)`, where *subsys* is the backend id.

**Called By:** db\_mcc\_setup  
initialize

Routines Called: none

Parameters: int subsys

**Returns:** `color_config_name[subsys]`

### 2.15.2.10 `set_database_name`

The `set_database_name` function sets the name of the terrain database file for a specified subsystem in the `database_name[]` array. This function is called when the Simulation Host

sends a MSG\_FILE\_DESCR message with *db\_req* set to DB\_SETUP\_0 or DB\_SETUP\_1.

The function call is **set database name(subsys, name)**, where:

***subsys*** is the backend id  
***name*** is the name of the file to be used

**Called By:** db\_mcc\_setup

**Routines Called:** strepy

```
Parameters:      int      subsys
                 char      *name
```

**Returns:** `none`

### 2.15.2.11 set\_ded\_name

The `set_ded_name` function sets the name of the Dynamic Elements Database file for a specified subsystem in the `ded_name[]` array. This function is called when the Simulation Host sends a `MSG_FILE_DESCR` message with `db_req` set to `DB_DED_SETUP_0` or `DB_DED_SETUP_1`.

The function call is set **ded name(subsys, name)**, where:

***subsys*** is the backend id  
***name*** is the name of the file to be used

**Called By:** db\_mcc\_setup

Routines Called: strcpy

```
Parameters:      int      subsys
                 char      *name
```

**Returns:** none

#### 2.15.2.12 set\_final\_lut\_name

The `set_final_lut_name` function sets the name of the final (2-D) lookup table file for a specified subsystem in the `final_lut_name[]` array. This function is called when the Simulation Host sends a `MSG_FILE_DESCR` message with `db_req` set to `DB_FINAL_LUT_0` or `DB_FINAL_LUT_1`.

The function call is **set\_final\_lut\_name(subsys, name)**, where:

*subsys* is the backend id

*name* is the name of the file to be used

Called By: db\_mcc\_setup

Routines Called: strcpy

Parameters: int subsys  
char \*name

Returns: none

#### 2.15.2.13 set\_3d\_lut\_name

The **set\_3d\_lut\_name** function sets the name of the 3-D lookup table file for a specified subsystem in the **lut\_3d\_name[]** array. This function is called when the Simulation Host sends a **MSG\_FILE\_DESCR** message with *db\_req* set to **DB\_3D\_LUT\_0** or **DB\_3D\_LUT\_1**.

The function call is **set\_3d\_lut\_name(subsys, name)**, where:

*subsys* is the backend id

*name* is the name of the file to be used

Called By: db\_mcc\_setup

Routines Called: strcpy

Parameters: int subsys  
char \*name

Returns: none

#### 2.15.2.14 set\_data\_2d\_name

The **set\_data\_2d\_name** function sets the name of the 2-D overlay data file for a specified subsystem in the **data\_2d\_name[]** array. This function is called when the Simulation Host sends a **MSG\_FILE\_DESCR** message with *db\_req* set to **DB\_2D\_DATA\_0** or **DB\_2D\_DATA\_1**.

The function call is **set\_data\_2d\_name(subsys, name)**, where:

*subsys* is the backend id  
*name* is the name of the file to be used

**Called By:** db\_mcc\_setup

Routines Called: `strcpy`

```
Parameters:      int      subsys
                  char      *name
```

**Returns:** none

### 2.15.2.15 set\_esifa\_name

The `set_esifa_name` function sets the name of the ESIFA textures file for a specified subsystem in the `esifa_name[]` array. This function is called when the Simulation Host sends a `MSG_FILE_DESCR` message with *db\_req* set to `DB_ESIFA_LOAD_0` or `DB_ESIFA_LOAD_1`.

The function call is **set\_esifa name(subsys, name)**, where:

*subsys* is the backend id  
*name* is the name of the file to be used

**Called By:** db\_mcc\_setup

Routines Called: strcpy

```
Parameters:      int      subsys
                 char      *name
```

**Returns:** none

### 2.15.2.16 `set_color_config_name`

The `set_color_config_name` function sets the name of the color configuration file for a specified subsystem in the `color_config_name[]` array. This function is called when the Simulation Host sends a `MSG_FILE_DESCR` message with `db_req` set to `DB_COLOR_CFG_0` or `DB_COLOR_CFG_1`.

The function call is **set\_color\_config name(subsys, name)**, where:

***subsys*** is the backend id

*name* is the name of the file to be used

Called By: db\_mcc\_setup

Routines Called: strcpy

Parameters: int subsys  
char \*name

Returns: none

## 2.16 System Utilities (/cig/libsrc/libutil)

The System Utilities CSC contains a set of utility functions used throughout the real-time software. These utilities provide the ability to do the following:

- Block-copy data.
- Open, read, and close file directories.
- Find fields in a file header.
- Find the latest version of a file with a given name.
- Convert S-format files into executable code.
- Perform VT100 graphics control.

Figure 2-20 identifies the CSUs in the System Utilities CSC.

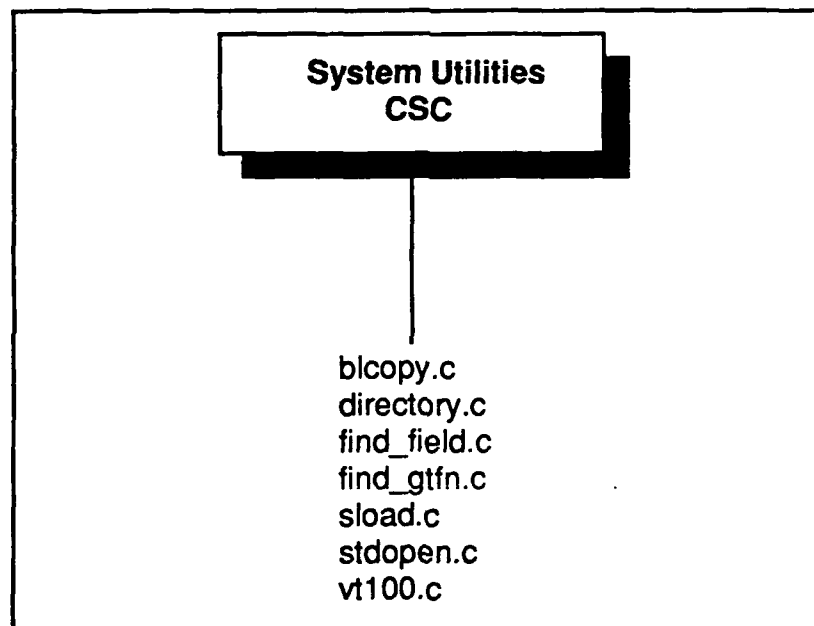


Figure 2-20. System Utilities CSUs

### 2.16.1 blcopy.c

The blcopy function block-copies a specified number of bytes from one memory location to another. For example, this function is used to copy data from AAM1 to AAM2.

The function call is **blcopy(from, to, count)**, where:

*from* is a pointer to the source location  
*to* is a pointer to the destination location  
*count* is the number of bytes to be copied

Called By: `_copy_reconfigurable_viewports_section`

\_move\_load\_module\_stp\_to\_quad\_buffer  
 \_update\_second\_active\_area\_memory  
 active\_area\_init  
 init\_simulation

Routines Called: none

Parameters:       int                               \*from  
                   int                               \*to  
                   int                               count

Returns:           none

### 2.16.2       directory.c

The functions in the directory.c CSU are used to open, read, and close file directories. These functions are:

- OpenDir
- ReadDir
- CloseDir
- GetFileName

#### 2.16.2.1    OpenDir

The OpenDir function opens a directory in read-only mode.

The function call is **OpenDir(dirname)**, where *dirname* is the name of the directory to be opened.

If successful, OpenDir returns the directory description as *desc*. If the directory could not be opened, OpenDir outputs an error and returns ERROR.

Called By:         find\_fn

Routines Called:   ifx\_open  
                      printf

Parameters:        char                            \*dirname

Returns:           desc  
                      -1 (ERROR)

### 2.16.2.2 ReadDir

The ReadDir function reads a specified entry from a specified directory.

The function call is **ReadDir(desc, pde)**, where:

*desc* is the directory description returned from OpenDir  
*pde* is a pointer to the entry to be read

The function returns RET\_OK if successful. It outputs an error and returns a 1 if the specified entry could not be read or has never been used.

Called By:	find_fn	
Routines Called:	ifx_read printf	
Parameters:	int IFXDIRENTRY	desc *pde
Returns:	0 (RET_OK) 1	

### 2.16.2.3 CloseDir

The CloseDir function closes an open directory.

The function call is **CloseDir(desc)**, where *desc* is the description returned by OpenDir for the directory.

Called By:	find_fn	
Routines Called:	ifx_close	
Parameters:	int	desc
Returns:	none	

### 2.16.2.4 GetFileName

The GetFileName function extracts a file name from a directory entry.



The function call is **GetFileName(pde, pfn)**, where:

*pde* is a pointer to the directory entry

*pfn* is a pointer to the file name

GetFileName puts the file name (including the extension, if any) in the location referenced by *pfn*.

Called By: find\_fn

Routines Called: none

Parameters: IFXDIRENTRY \*pde  
char \*pfn

Returns: none

### 2.16.3 find\_field.c (FindField)

The FindField function locates a specified header descriptor and returns its associated data field. This function is used by mpvideo\_load to get information such as the file type and file length from the GSP header.

The function call is **FindField(ss, buf)**, where:

*ss* is the field name to be found

*buf* is the header descriptor to be searched

FindField uses search to find the specified field, then parses it to find the data. It returns the data as *ps*. *ps* is set to NULL if the field cannot be found.

Called By: mpvideo\_load

Routines Called: search

Parameters: char \*ss  
char \*buf

Returns: ps

#### 2.16.4 find\_gtn.c (find\_fn)

The find\_fn function finds the file that has the highest extension and whose name matches a given character string. This ensures that the calling procedure loads the latest version of a file.

The function call is find\_fn(compare, size, exact, filename), where:

*compare* is the name to be matched

*size* is the number of characters in the compare string

*exact* specifies whether or not the file name must match the compare string exactly

*filename* is a pointer to the file name found by find\_fn

find\_fn does the following:

- If an exact match is specified:
  - Converts the compare string to uppercase.
  - Tries to open a file with that name.
  - If a match is found, puts the name in *filename*, outputs a message, and closes the file.
  - If no match is found, outputs a message and continues to search as if an exact match had not been specified.
- If an exact match is not specified, or an exact match cannot be found:
  - Converts the compare string to uppercase.
  - Finds the directory name in the compare string.
  - Calls opendir to open the directory.
  - Allocates memory for a directory entry block to work in.
  - Calls readdir to read each directory entry, looking for matching file names.
  - If a match is found, checks to see if the extension is greater than any previous match found.
  - If the extension is greater than any previous match, extracts the file name (by calling GetFileName) and saves it.
  - When all directory entries have been read, calls closedir to close the directory.
  - Frees the directory entry memory block.
  - If a match was found, puts the name in *filename* and outputs a message.

The function returns TRUE if a match is found, or FALSE if no match is found. It returns ERROR if the directory could not be opened or the working directory could not be allocated.

Called By:           flea\_init\_cig\_sw  
                      gos\_120tx  
                      initialize  
                      open\_dbase  
                      open\_ded

Routines Called:    calloc  
                      close  
                      Closedir

free  
 GetFileName  
 open  
 OpenDir  
 printf  
 ReadDir  
 strcat  
 strcmp  
 strcpy  
 strlen  
 strncmp  
 strncpy  
 toupper

Parameters:	char int BOOLEAN char	*compare size exact *filename
Returns:	1 (TRUE) 0 (FALSE) -1 (ERROR)	

### 2.16.5 sload.c

The functions in the sload.c CSU are used to convert a Motorola S-format file into executable code. These functions are:

- sload
- get\_record
- send\_data
- check\_sum
- get\_binary\_data
- get\_char
- ctoi

#### 2.16.5.1 sload

The sload function converts a Motorola S-format file into executable code. It reads data from the disk in sector-sized chunks, breaks the ASCII down into record-sized lines, then stores the binary data. This function is used to load the forcetask file.

The function call is **sload(filename, offset, wsize)**, where:

*filename* is the file to be converted  
*offset* is the amount to add to the binary data address  
*wsize* is the size of the destination granularity: c (character) or w (word)

sload does the following:

- Validates the value specified for *wsize*.
- Calls `open` to open the specified file.
- Calls `get_record` to get each record from the file.
- Call `check_sum` to perform a checksum on each record.
- If the record is S7, S8, or S9, closes the file and returns OK.
- If the record is S0 or S5, ignores it.
- If the record is S1, S2, or S3, calls `get_binary_address` and `send_data` to collect the addresses.

The function returns OK if successful. It returns ERROR if the write size is invalid, the file could not be opened, a record has a bad checksum, or an early end-of-file is detected.

Called By:	bootforce	
Routines Called:	check_sum close get_binary_data get_char get_record open printf send_data	
Parameters:	char INT_4 char	*filename offset wsize
Returns:	0 (OK) -1 (ERROR)	

#### 2.16.5.2 get\_record

The `get_record` function fills a string buffer with exactly one Motorola S-format record.

The function call is `get_record(record)`, where *record* is the record to be read. `get_record` does the following:

- Calls `get_char` to get the first character from the record.
  - If the character is 'S' (the leading character), ignores it.
  - If the character is 'EOF' (end-of-file), returns a 0 and stops.
  - If the character is anything else, stores it.
- Calls `get_char` to read the next three characters (the record type and byte count).
- Calls `get_binary_data` to get the byte count.
- Calls `get_char` to read the remaining bytes (based on the byte count).

The function returns the S-format byte count if successful. It returns 0 if there are no records in the file.

Called By:	sload	
Routines Called:	get_binary_data get_char printf	
Parameters:	BYTE	record[]
Returns:	0 byte_count	

### 2.16.5.3 send\_data

The `send_data` function writes ASCII record data to memory in ascending bytes from a given start address.

The function call is `send_data(address, cptr, count, wsize)`, where:

*address* is the initial load address (absolute S-format)  
*cptr* is a pointer to the ASCII record characters  
*count* is the number of characters to transmit  
*wsize* is the size of the destination granularity: **c** (character) or **w** (word)

Called By:	sload	
Routines Called:	get_binary_data	
Parameters:	WORD char INT_4 char	address *cptr count wsize
Returns:	none	

### 2.16.5.4 check\_sum

The `check_sum` function verifies the checksum byte of an S-format record.

The function call is `check_sum(pointer, count)`, where:

*pointer* points to the record to be checksummed  
*count* is the byte count

The *answer* returned by the function is 0 if the checksum byte is correct. A non-zero value indicates a bad checksum.

Called By:           sload

Routines Called:     get\_binary\_data

Parameters:          char                           \*pointer  
                      INT\_4                        count

Returns:            answer

#### 2.16.5.5     get\_binary\_data

The get\_binary\_data function receives a pointer to a character string and returns the binary equivalent of the first specified number of characters.

The function call is **get\_binary\_data(cptr, count)**, where:

*cptr* is a pointer to the character string  
*count* is the number of characters (starting from the beginning of the string) to be converted

The result is returned as *binary\_data*.

Called By:           check\_sum  
                      get\_record  
                      send\_data  
                      sload

Routines Called:     ctoi

Parameters:          char                           \*cptr  
                      INT\_4                        count

Returns:            binary\_data

#### 2.16.5.6     get\_char

The get\_char function returns the next available ASCII character from a sector-sized buffer. If the buffer is empty, get\_char reads the next sector from disk. If there is no next sector, get\_char returns EOF.

The function call is **get\_char()**.

Called By: get\_record

Routines Called: read

Parameters: none

Returns: \*bptr++  
EOF

#### 2.16.5.7 ctoi

The ctoi function converts a character to an integer, and returns the integer.

The function call is **ctoi(c)**, where *c* is the character to be converted.

Called By: get\_binary\_data

Routines Called: none

Parameters: char c

Returns: c - '0'  
c - 'A' + 10

#### 2.16.6 stdopen.c

The stdopen function opens the console as stdin (in read-only mode), stdout (in write-only mode), and stderr (in write-only mode).

The function call is **stdopen()**.

This function is not currently used.

Called By: none

Routines Called: ifx\_open

Parameters: none

Returns: none

### 2.16.7 vt100.c

The functions in the vt100.c CSU provide VT100 graphics control. These function are used primarily by the Flea and Gossip menu handlers. The functions are:

- cup
- sgr
- double\_top
- double\_bot
- double\_off
- blank
- save\_cur
- restore\_cur
- scroll\_reg

#### 2.16.7.1 cup

The cup function positions the cursor at a specified row and column.

The function call is **cup(r, c)**, where:

*r* is the row  
*c* is the column

Called By:

derror  
flea\_agpt\_locations  
flea\_agpt\_locations\_main\_menu  
flea\_agpt\_switches  
flea\_agpt\_switches\_main\_menu  
flea\_atp  
flea\_atp\_main\_menu  
flea\_bal\_opts  
flea\_bal\_opts\_main\_menu  
flea\_draw\_2d  
flea\_graphics\_test  
flea\_graphics\_test\_main\_menu  
flea\_prompt  
flea\_switches  
flea\_switches\_main\_menu  
flea\_veh\_control  
flea\_veh\_control\_main\_menu  
gos\_db\_query  
gos\_db\_query\_menu  
gos\_main\_menu  
gos\_ppm\_query  
gos\_ppm\_query\_menu  
gos\_prompt  
gossip\_tick  
host\_if\_debug\_main\_menu  
host\_if\_debug\_menu



host\_if\_disable\_debug\_msgs  
 host\_if\_display\_enabled\_msgs  
 host\_if\_enable\_debug\_msgs  
 host\_list\_msgs  
 menu\_header  
 model\_demo  
 tick  
 tick\_main\_menu  
 tick\_ppm  
 tick\_ppm\_menu  
 tick\_ppm\_menu\_header  
 tick\_ppm\_update\_info  
 tick\_script  
 tick\_script\_main\_menu  
 update\_menu\_header

Routines Called:     printf

Parameters:         INT\_4                     r  
                       INT\_4                     c

Returns:             none

#### 2.16.7.2     sgr

The sgr function is used for special graphics renditions.

The function call is sgr(v), where v is the row number.

This function is not currently used.

Called By:           none

Routines Called:     printf

Parameters:         INT\_4                     v

Returns:             none

#### 2.16.7.3     double\_top

The double\_top function represents double-wide, double-high for the top half of the monitor screen.

The function call is double\_top(s), where s is the starting line.

This function is not currently used.

Called By: none

Routines Called: printf

Parameters: BYTE s

Returns: none

#### 2.16.7.4 double\_bot

The double\_bot function represents double-wide, double-high for the bottom half of the monitor screen.

The function call is **double\_bot(s)**, where *s* is the starting line.

This function is not currently used.

Called By: none

Routines Called: printf

Parameters: BYTE s

Returns: none

#### 2.16.7.5 double\_off

The double\_off function returns to single-high and single-width. This reverses the effect of double\_top and/or double\_bot.

The function call is **double\_off()**.

This function is not currently used.

Called By: none

Routines Called: printf

Parameters: none

Returns: none

#### 2.16.7.6 blank

The blank function clears the screen from a given starting position.

The function call is **blank(m)**, where *m* is the character position from which the screen is to be blanked.

Called By:

- flea\_agpt\_locations\_main\_menu
- flea\_agpt\_switches\_main\_menu
- flea\_atp\_main\_menu
- flea\_bal\_opts
- flea\_bal\_opts\_main\_menu
- flea\_graphics\_test\_main\_menu
- flea\_switches
- flea\_switches\_main\_menu
- flea\_veh\_control
- flea\_veh\_control\_main\_menu
- gos\_db\_query
- gos\_db\_query\_menu
- gos\_main\_menu
- gos\_ppm\_query
- gos\_ppm\_query\_menu
- gossip\_tick
- host\_if\_debug\_main\_menu
- host\_if\_display\_enabled\_msgs
- host\_list\_msgs
- model\_demo
- tick
- tick\_main\_menu
- tick\_ppm
- tick\_ppm\_menu
- tick\_script
- tick\_script\_main\_menu

Routines Called: printf

Parameters: INT\_4 m

Returns: none

#### 2.16.7.7 save\_cur

The save\_cur function saves the current cursor position.

The function call is **save\_cur()**.

This function is not currently used.

Called By: none

Routines Called: printf

Parameters: none

Returns: none

#### 2.16.7.8 restore\_cur

The **restore\_cur** function restores the cursor position to the location saved by **save\_cur**.

The function call is **restore\_cur()**.

This function is not currently used.

Called By: none

Routines Called: printf

Parameters: none

Returns: none

#### 2.16.7.9 scroll\_reg

The **scroll\_reg** function sets the top and bottom boundaries of the scrolling region.

The function call is **scroll\_reg(t, b)**, where:

*t* is the top of the scroll region  
*b* is the bottom of the scroll region

This function is not currently used.

Called By: none

Routines Called:    printf

Parameters:        INT\_4                            t  
                     INT\_4                           b

Returns:            none

## 2.17 Viewport Configuration (/cig/libsrc/libvpt)

The Viewport Configuration CSC is responsible for initializing and building the configuration tree before runtime. The configuration tree describes the relationship between the physical components of the simulated vehicle and the location of the viewports. The data used to set up the configuration tree is input via messages received from the Simulation Host.

The configuration tree consists of the following:

- One root node, which marks the start of the configuration tree. This node contains no data and must be the first node created.
- One or more matrix nodes, each of which contains a transformation matrix that specifies rotation angles (heading, pitch, and roll) and translation values. The matrices in all nodes in a traversal path of the tree are concatenated to generate the view of the world for the viewport represented by that path. Matrix nodes are designated as either dynamic (ones that are updated during the simulation) or static (ones that do not change during the simulation).
- Zero or more conditional (branch) nodes, each of which branch into one of two traversal paths based on a runtime condition. The node branched to if the condition is true is the conditional node's "true child" and the node branched to if the condition is false is the "false child." The branch values are stored in the system view flags array. The branch values in effect at any given time in the simulation are set via messages sent from the Simulation Host.
- Viewport parameters for each viewport. These parameters are the screen resolution, viewing range, near plane, field-of-view angles, level-of-detail multiplier, and aspect ratio (currently not used). Viewport parameters are associated with the final node in each traversal path in the configuration tree.

Note that the same viewport may be defined multiple times, each with different parameters. A conditional node enables a change to new viewport parameters during the simulation.

- One or more sets of graphics path parameters for each viewport. A graphics path is a window on a viewport. On a T backend, there is one graphics path per viewport. On a TX backend, there are two or four graphics paths per viewport, depending on the resolution. The graphics path parameters are used to load the hardware.

The structure of the configuration tree cannot be changed during runtime — all nodes and viewport definitions must be created at CIG initialization time. However, various parameters *within* the configuration tree do change during the simulation. Therefore, some Viewport Configuration functions are called to update configuration tree structures during runtime.

Specifically, messages can be used to update the following structures after the configuration tree has been created:

- Dynamic matrices. The Simulation Host can provide a new matrix or a change (e.g., rotation) to the current matrix.

- Branch values for conditional nodes. Changing the branch values during a simulation causes selection of a different traversal path and, typically, different viewport parameters.
- Certain viewport parameters (the level-of-detail multiplier and the field-of-view angles). Although a message is available to change these parameters directly, it is recommended that all desirable viewport parameter combinations be built into the configuration tree and selected using branch values.

The configuration tree can contain a maximum of 64 nodes. Each node is referenced by a unique index, which is used in messages sent to update the node during the simulation. The root node is always assigned node\_index 0. A node that has viewport parameters attached to it must have a node\_index between 1 and 31.

Every matrix node in the configuration tree must be defined in one of two formats: RTS4x3 (4 x 3 rotation translation scale) or HPRXYZS (3 x 3 scale heading pitch roll translation). A matrix node's format can be redefined during the simulation.

The format of each of these matrix structures is as follows:

#### RTS4x3 (4 x 3 rotation translation scale)

The matrix format is:

rotation[0,0]	rotation[0,1]	rotation[0,2]
rotation[1,0]	rotation[1,1]	rotation[1,2]
rotation[2,0]	rotation[2,1]	rotation[2,2]
translation.x	translation.y	translation.z

where:

*rotation* is an angle in degrees

*translation* is a distance in meters

The typedef for this matrix structure is:

```
typedef struct {
    REAL_4      rotation[3][3];
    R4P3D      translation;
} RTS4x3_MTX;
```

#### HPRXYZS (3 x 3 scale heading pitch roll translation)

The matrix format is:

heading	pitch	roll
translation.x	translation.y	translation.z
scale.x	scale.y	scale.z
scale order	heading order	pitch order
roll order	translate order	

where:

*heading* = -yaw = -z rotation in degrees

*pitch* = x rotation in degrees

*roll* = y rotation in degrees

*translation* is a distance in meters

*scale* is a scaling factor (used to enlarge or reduce matrices)  
the *order* values specify the order in which the matrices are to be concatenated

The typedef for this matrix structure is:

```
typedef struct {
    REAL_4      heading;
    REAL_4      pitch;
    REAL_4      roll;
    R4P3D       translation;
    R4P3D       scale;
    UNS_1       concat_order[5];
} RTS3x3_MTX;
```

A third matrix format, **ROT2x1 (2 x 1 rotation)**, can be used to rotate a matrix along one axis. Matrix nodes cannot be defined as this matrix format, although they can be updated by it. The matrix format for ROT2x1 is:

$\begin{bmatrix} \cos(\text{rotation } ^\circ) & \sin(\text{rotation } ^\circ) \\ \text{rotation axis} \end{bmatrix}$
---

where:

*rotation* is the angle of rotation in degrees

*rotation axis* is the axis along which rotation is to occur: 0 (x), 1 (y), or 2 (z)

The typedef for this matrix structure is:

```
typedef struct {
    REAL_4      cos_rotation;
    REAL_4      sin_rotation;
    UNS_1       rotation_axis;
    UNS_1       unused[3];
} ROT2x1_MTX;
```

The functions in the Viewport Configuration CSC do the following:

- Create all configuration nodes, viewport parameter entries, and graphics path entries, based on data received from the Simulation Host.
- Generate DTP-style matrices from the matrices provided by the Simulation Host.
- Generate DTP code for the overlays.
- Process the system view flags/branch values and load them into the T&C (Timing and Control) board.
- Provide an interface to the Gossip user to test the viewport configuration process without connection to a Simulation Host.

The configuration tree is built according to messages received from the Simulation Host. To initiate this process, `db_mcc_setup` (in the Real-Time Processing CSC) calls the



`cig_config` function (in the CIG Configuration CSC). `cig_config` in turn calls the Viewport Configuration functions to configure the nodes, viewports, and graphics paths.

Most of the CSUs in the Viewport Configuration CSC fall into one of the following groups:

**`cnode_*` CSUs**

Contain functions that operate on configuration nodes.

**`vpt_*` CSUs**

Contain functions that operate on nodes that have viewport parameters attached to them.

**`mtx_*` CSUs**

Contain functions that perform matrix-specific operations.

**`path_*` CSUs**

Contain functions that operate on graphics path parameters.

**`tst_*` CSUs**

Contain functions useful for testing the viewport configuration process. These include a routine that can be used to create the tree from data in an ASCII file created offline.

**`u_*` CSUs**

Contain the update functions used to actually modify the viewport objects. These routines are called by the other Viewport Configuration CSUs.

Many of the function names are preceded with `vpt_` followed by the CSU type. For example, most of the functions that deal with configuration nodes begin with `vpt_cnode`. Similarly, most of the update functions begin with `vpt_update`.

Figure 2-21 identifies the CSUs in Viewport Configuration. The functions performed by these CSUs are described in this section.

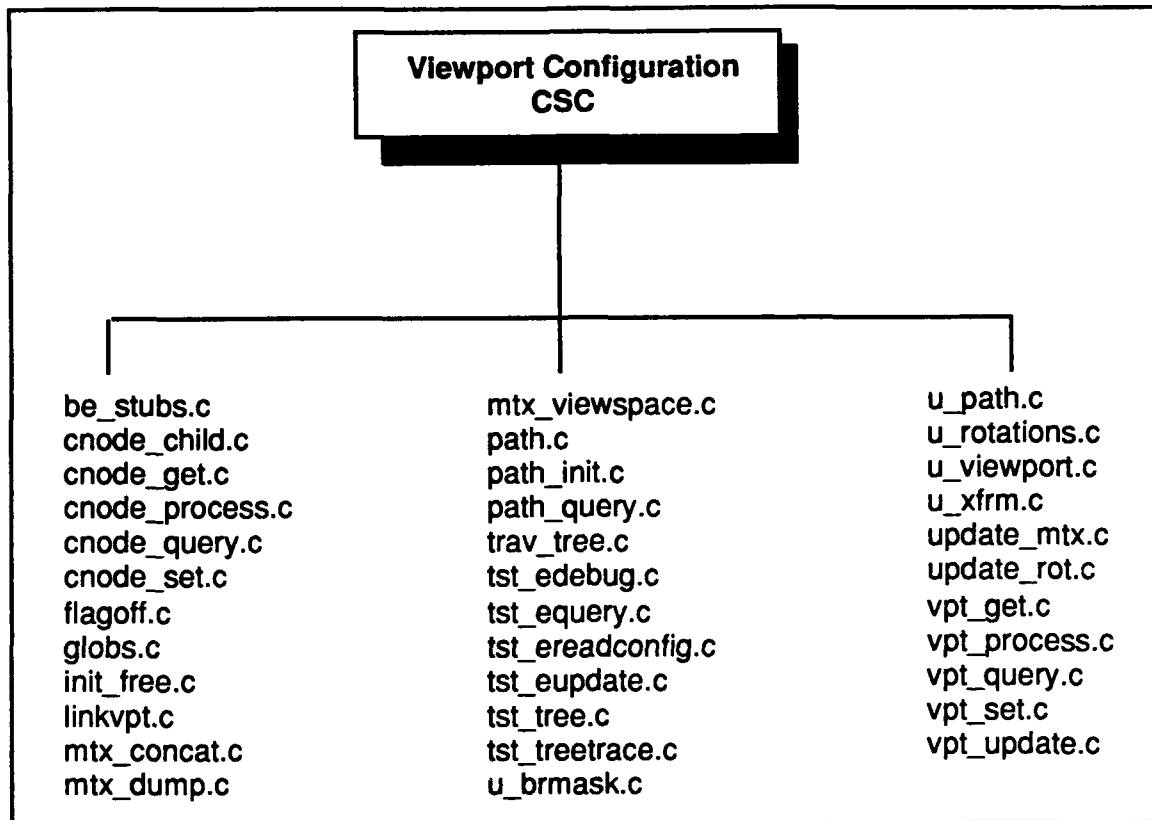


Figure 2-21. Viewport Configuration CSUs

### 2.17.1 be\_stubs.c

The functions in the `be_stubs.c` CSU are used to interface to the backend. These functions are the following:

- `be_query_num_paths`
- `find_be_id`
- `be_query_buffer_offset`
- `be_qulm`
- `be_query_lm_per_lmb_side`
- `vpt_init_mode_on`
- `vpt_init_mode_off`
- `aam_free`

#### 2.17.1.1 be\_query\_num\_paths

The `be_query_num_paths` function determines the number of graphics path required to build a viewport. This function is called when viewport parameters are created for configuration nodes.

The function call is `be_query_num_paths(vpt_id, n_vpts, res_i, res_j, num_pi, num_pj, start_p)`, where:

*vpt\_id* is the viewport identifier

*n\_vpts* is the number of viewports

*res\_i* is the number of pixels in each row (horizontal)

*res\_j* is the number of pixels in each column (vertical)

*num\_pi* is a pointer to the number of paths in the i direction

*num\_pj* is a pointer to the number of paths in the j direction (currently always 1)

*start\_p* is a pointer to the starting path id

*be\_query\_num\_paths* does the following:

- Calls *find\_be\_id* to get the viewport's backend id.
- Calls *backend\_get\_object\_addr* to get the address of the backend.
- For a TX backend, sets the number of graphics paths to 1, 2, or 4 based on the desired resolution and the number of viewports.
- For a T backend, sets the number of graphics paths to 1.
- Determines the start path id based on which backend the viewport is in, the type of backend, and the resolution.
- Sets the remaining values.

The function returns SUCCEED if it is successful. It returns FAIL if the backend identifier is invalid or the backend could not be found.

Called By: *vpt\_vpt\_process*

Routines Called: *backend\_get\_object\_addr*  
*find\_be\_id*  
*printf*

Parameters:	INT_2	<i>vpt_id</i>
	INT_2	<i>n_vpts</i>
	INT_4	<i>res_i</i>
	INT_4	<i>res_j</i>
	INT_2	<i>*num_pi</i>
	INT_2	<i>*num_pj</i>
	INT_2	<i>*start_p</i>

Returns: 1 (SUCCEED)  
0 (FAIL)

### 2.17.1.2 *find\_be\_id*

The *find\_be\_id* function, given a viewport id, returns the corresponding backend id. It returns 0 if the viewport id is between 0 and 7, 1 if the viewport id is between 8 and 15, and -1 (an error) if the viewport id is anything else.

The function call is *find\_be\_id(vpt)*, where *vpt* is the viewport id.

Called By: *be\_query\_num\_paths*



Routines Called: none

Parameters: none

Returns: DB0

### 2.17.1.5 be\_qulm

The `be_qulm` function returns database-specific load module data for a specified viewport. The values it returns are the the inverse of the load module block width and the number of load module blocks per side of active area memory.

The function call is `be_qulm(vpt_id, inv_lmb_width, lmb_per_side)`, where:

*vpt\_id* is the viewport id

*inv\_lmb\_width* is a pointer to the inverse of the width of a load module block

*lmb\_per\_side* is a pointer to the number of load module blocks per side of AAM

`be_qulm` determines which backend the viewport is in, then puts the data into the locations passed in the call.

Called By: dtp\_trav2

Routines Called: none

Parameters:	INT_2	vpt_id
	REAL_4	*inv_lmb_width
	REAL_4	*lmb_per_side

Returns: none

### 2.17.1.6 be\_query\_lm\_per\_lmb\_side

The `be_query_lm_per_lmb_side` function returns the number of load modules on a side of a load module block for a given database. This value is used for scaling when load module blocking is enabled. Usually, the value is 1 for 3500-meter (non-blocked) databases and 2 for 7000-meter (blocked) databases.

The function call is `be_query_lm_per_lmb_side(vpt_id)`, where *vpt\_id* is the viewport id.

The function determines which database to query based on the viewport id, then returns the result as *temp*.

Called By:           vpt\_vpt\_process

Routines Called:     none

Parameters:          INT\_2                           vpt\_id

Returns:             temp

#### 2.17.1.7    vpt\_init\_mode\_on

The `vpt_init_mode_on` function sets the viewport initialization mode (`vpt_init_mode`) variable to TRUE. This variable indicates whether or not viewport configuration is in progress. `vpt_init_mode_on` is called by `cig_config` before it starts building the configuration tree.

The function call is `vpt_init_mode_on(backend)`, where *backend* is the id of the backend to be initialized.

The function also sets the `init_backend` variable to the specified backend.

Called By:           cig\_config

Routines Called:     none

Parameters:          INT\_4                           backend

Returns:             none

#### 2.17.1.8    vpt\_init\_mode\_off

The `vpt_init_mode_off` function sets the viewport initialization mode (`vpt_init_mode`) variable to FALSE. This variable indicates whether or not viewport configuration is in progress. `vpt_init_mode_off` is called by `cig_config` after the configuration tree has been built and the DTP compiler has finished.

The function call is `vpt_init_mode_off()`.

Called By:           cig\_config

Routines Called:     none

Parameters: none

Returns: none

#### 2.17.1.9 aam\_free

The `aam_free` function is a dummy function that performs no action. This function is used to <<TBD>>.

The function call is `aam_free(ptr)`, where *ptr* is the location to be freed.

Called By: `vpt_tree_free`

Routines Called: none

Parameters: `UNS_1` \*ptr

Returns: none

#### 2.17.2 cnode\_child.c

The functions in the `cnode_child.c` CSU are used to add branch and standard (matrix) child nodes to the configuration tree at initialization time. These functions are:

- `vpt_cnode_set_bchild`
- `vpt_cnode_set_stdchild`

##### 2.17.2.1 vpt\_cnode\_set\_bchild

The `vpt_cnode_set_bchild` function adds branch child nodes (the children of branch nodes) to the configuration tree.

The function call is `vpt_cnode_set_bchild(new_node, parent_node, true_child_flag)`, where:

*new\_node* is a pointer to the new child node

*parent\_node* is a pointer to the child's parent node

*true\_child\_flag* indicates whether this is the branch node's true child or the false child

`vpt_cnode_set_bchild` does the following:

- For a true child, puts the pointer to the child in the parent's true child slot, or outputs an error if this slot is already filled.
- For a false child, puts the pointer to the child in the parent's false child slot, or outputs an error if this slot is already filled.

Called By: vpt\_cnode\_process

Routines Called: printf  
vpti\_state\_dcnode (in debug mode only)

Parameters: CONFIGURATION\_NODE \*new\_node  
CONFIGURATION\_NODE \*parent\_node  
UNS\_1 true\_child\_flag

Returns: none

### 2.17.2.2 vpt\_cnode\_set\_stdchild

The vpt\_cnode\_set\_stdchild adds matrix child nodes (the children of matrix nodes) to the configuration tree.

The function call is **vpt\_cnode\_set\_stdchild(new\_node, parent\_node)**, where:

*new\_node* is a pointer to the new child node  
*parent\_node* is a pointer to the child's parent node

vpt\_cnode\_set\_stdchild does the following:

- If the parent's first child slot is empty, (i.e., the parent has no other children), puts a pointer to the child in that slot.
- If the parent's first child slot is filled (i.e., the parent has at least one other child), traverses down the parent node's sibling list. Puts a pointer to the new node in the sibling pointer slot of the last sibling.

Called By: vpt\_cnode\_process

Routines Called: printf (in debug mode only)  
vpti\_state\_dcnode (in debug mode only)

Parameters: CONFIGURATION\_NODE \*new\_node  
CONFIGURATION\_NODE \*parent\_node

Returns: none

### 2.17.3 cnode\_get.c (vpt\_cnode\_get)

The vpt\_cnode\_get function allocates memory for a configuration node and returns a pointer to that node. This function is called by vpt\_cnode\_process each time it adds a new node to the configuration tree.



The function call is **vpt\_cnode\_get(*node\_index*)**, where *node\_index* is the index of the node for which a pointer is to be returned.

**vpt\_cnode\_get** does the following:

- Verifies that the *node\_index* is valid and the array can hold another pointer. Outputs an error if the node cannot be added.
- Calls **vpti\_get\_ptr\_cnode** to make sure this node does not already have a pointer set to it. Outputs an error if it has.
- Calls **calloc** to allocate memory for this configuration node.
  - If memory is available, calls **vpti\_set\_ptr\_cnode** to add the pointer to the node pointers allocation table (**Gvpt\_cnode\_pointers**).
  - If not enough memory is available, outputs an error.

The function returns the pointer assigned to the node as *cnode*.

Called By: **vpt\_cnode\_process**

Routines Called: **calloc**  
**printf**  
**vpti\_get\_ptr\_cnode**  
**vpti\_set\_ptr\_cnode**

Parameters: **INT\_2** **node\_index**

Returns: **cnode**

#### 2.17.4 **cnode\_process.c (vpt\_cnode\_process)**

The **vpt\_cnode\_process** function is responsible for placing all configuration node information into the configuration tree. This function is called by **cig\_config** when the Simulation Host sends a **MSG\_CREATE\_CONFIGNODE** message. Based on the node's type, **vpt\_cnode\_process** calls the appropriate function to add the new node.

The function call is **vpt\_cnode\_process(*node\_index*, *parent\_index*, *node\_type*, *matrix\_type*, *true\_child\_flag*, *static\_mtx\_flag*, *branch\_index*, *branch\_mask*, *matrix*, *description*)**, where:

*node\_index* is the unique index used to reference this node

*parent\_index* is the *node\_index* of this node's parent node

*node\_type* is **0** (matrix node), **1** (branch node), or **2** (branch/matrix node — for future expansion)

*matrix\_type* is **0** (HPRXYZS matrix) or **1** (RTS4x3 matrix); matrices cannot be defined as **2** (ROT2x1 matrix)

*true\_child\_flag* is **0** (false child of a branch node) or **1** (true child of a branch node)

*static\_mtx\_flag* is **0** (dynamic node, matrix will move in runtime) or **1** (static node, matrix will not change during the simulation)

*branch\_index* is the node's index into the branch value array (for branch nodes only)

*branch\_mask* is the node's branch mask (branch values for a branch node, values controlling overlay placement for a matrix node)  
*matrix* is the node's initial rotation matrix  
*description* is a text string describing the node

vpt\_cnode\_process does the following:

- Calls vpt\_cnode\_get to get a pointer to the new node.
- Copies the node index, type, and description to the new node entry.
- If this is the root node (node\_index = 0):
  - Resets the vehicle\_id.
  - Calls vpt\_root\_init to initialize the root node, view positions array, and view flags array.
  - Returns the status returned by vpt\_root\_init.
- If this is a child node:
  - Loads the parent index into the node entry.
  - Calls vpti\_get\_ptr\_cnode to get the pointer for the parent node, and loads it into the node entry.
  - If the parent is a branch node, calls vpt\_cnode\_set\_bchild to add the new child node.
  - If the parent is a matrix node, calls vpt\_cnode\_set\_stdchild to add the new child node.
- If this is a matrix node:
  - Calls vpt\_cnode\_set\_matrix to generate the node's matrix and load it into active area memory.
- If this is a branch (conditional) node:
  - Calls vpt\_cnode\_set\_branch to add the node's branch index and branch mask, and to set a pointer to the view flags/branch value array.
- If this is a child of the root node (i.e., a world/hull matrix node):
  - Assigns the next vehicle id and sets it in the node entry.
  - Calls process\_vppos to load the vehicle's current position.

The function returns SUCCEED if the node is added to the configuration tree successfully. It returns FAIL if memory could not be allocated for the new node, or if one of the called routines could not add the node for any reason.

Called By:            cig\_config  
                       p\_configtree\_node

Routines Called:    printf  
                       process\_vppos  
                       strcpy  
                       vpt\_cnode\_get  
                       vpt\_cnode\_set\_bchild  
                       vpt\_cnode\_set\_branch  
                       vpt\_cnode\_set\_matrix  
                       vpt\_cnode\_set\_stdchild  
                       vpt\_root\_init  
                       vpti\_get\_ptr\_cnode  
                       vpti\_state\_dcnode    (in debug mode only))

Parameters:	INT_2	node_index
	INT_2	parent_index
	UNS_1	node_type
	UNS_1	matrix_type
	UNS_1	true_child_flag
	UNS_1	static_mtx_flag
	UNS_1	branch_index
	UNS_4	branch_mask
	MTXUNION	*matrix
	UNS_1	description[]

Returns: 1 (SUCCEED)  
0 (FAIL)

### 2.17.5 cnode\_query.c

The functions in the cnode\_query.c CSU are used to read configuration node data. These functions are:

- vpt\_cnode\_query
- vpt\_cnode\_qroot

#### 2.17.5.1 vpt\_cnode\_query

The vpt\_cnode\_query function copies information from the configuration node structure to a query structure. This function is used in testing only.

The function call is **vpt\_cnode\_query(cp, ci)**, where:

*cp* is a pointer to the configuration node  
*ci* is a pointer to the information structure

vpt\_cnode\_query does the following:

- Initializes the query information structure.
- Puts all data from the configuration node (node index, parent index, matrix type, etc.) into the query information structure.

Called By:	tst_tretrace vptq_brvals vptq_cnout vptq_dynmtx
------------	--

Routines Called:	none
------------------	------

Parameters:	CONFIGURATION_NODE	*cp
	VPT_CNODE_INFO	*ci

Returns: none

### 2.17.5.2 vpt\_cnode\_qroot

The `vpt_cnode_qroot` function returns a pointer to the root node, which it finds by calling `vpti_get_ptr_cnode`.

The function call is `vpt_cnode_qroot()`.

Called By: dtp\_compiler  
process\_vflags  
process\_vpops  
vptq\_activept

Routines Called: vpti\_get\_ptr\_cnode

Parameters: none

Returns: vpti\_get\_ptr\_cnode(0)

### 2.17.6 cnode\_set.c

The functions in the `cnode_set.c` CSU are used to add branch and matrix node parameters to the configuration tree. These functions are:

- `vpt_cnode_set_branch`
- `vpt_cnode_set_matrix`

#### 2.17.6.1 vpt\_cnode\_set\_branch

The `vpt_cnode_set_branch` function sets the special parameters that pertain only to branch (conditional) nodes. This function is called by `vpt_cnode_process` when adding a branch node to the configuration tree.

The function call is `vpt_cnode_set_branch(new_node, branch_mask, branch_index)`, where:

*new\_node* is a pointer to the new node's entry in the configuration tree  
*branch\_mask* is the node's branch mask  
*branch\_index* is the index into the branch value array

`vpt_cnode_set_branch` does the following:

- Puts the node's branch mask into the node entry.

- Determines the node's branch value pointer by using its node index to index into the branch value array. (The address of this array is stored in the root node's branch value pointer.)
- Puts the node's branch value pointer into the node entry.

Called By: `vpt_cnode_process`

Routines Called: `printf` (in debug mode only)  
`vpti_get_ptr_cnode`  
`vpti_state_dcnode` (in debug mode only)

Parameters: `CONFIGURATION_NODE*` `new_node`  
`UNS_4` `branch_mask`  
`UNS_1` `branch_index`

Returns: `none`

#### 2.17.6.2 `vpt_cnode_set_matrix`

The `vpt_cnode_set_matrix` function sets the special parameters required for matrix nodes, generates the DTP matrix, and loads the matrix into active area memory. This function is called by `vpt_cnode_process` when adding a matrix node to the configuration tree.

The function call is `vpt_cnode_set_matrix(new_node, matrix_type, static_mtx_flag, matrix, branch_mask)`, where:

*new\_node* is a pointer to the new node entry  
*matrix\_type* is `0` (HPRXYZS matrix) or `1` (RTS4x3 matrix)  
*static\_mtx\_flag* is `0` (dynamic node, matrix will move in runtime) or `1` (static node, matrix will not change during the simulation)  
*matrix* is the node's initial rotation matrix  
*branch\_mask* is the node's branch mask (used to indicate whether or not overlays are to be displayed on the viewport)

`vpt_cnode_set_matrix` does the following:

- Puts the matrix type and static matrix flag into the new node's entry.
- Allocates memory for the matrix in the node.
- Allocates memory for the DTP matrix in active area memory.
- Makes a copy of the matrix.
- Allocates memory for a copy of the matrix (for dynamic matrices only), for use in terrain feedback processing.
- Calls `concat_mtx` to generate the DTP-style matrix and load it into active area memory.
- Puts the node's branch mask into the node entry.

The function returns `SUCCESS` if the matrix data was generated successfully. It returns `FAIL` if memory could not be allocated for the node matrix, the DTP matrix, or the copy of the dynamic matrix.

Called By: vpt\_cnode\_process

Routines Called: aam\_malloc  
 calloc  
 concat\_mtx  
 mtxcpy  
 printf  
 r4mat\_dump (in debug mode only)  
 vpti\_state\_dcnode (in debug mode only)

Parameters: CONFIGURATION\_NODE\* new\_node  
 UNS\_1 matrix\_type  
 UNS\_1 static\_mtx\_flag  
 UNS\_4 branch\_mask  
 MTXUNION \*matrix

Returns: 1 (SUCCEED)  
 0 (FAIL)

### 2.17.7 flagoff.c

The flagoff function turns off all Viewport Configuration debug flags. This function is called by cig\_config (in the CIG Configuration CSC) when it processes the CIG Control-Stop message after the configuration tree has been built.

The function call is **flagoff()**. flagoff sets Gvpt\_dtree\_short to TRUE and sets the following variables to FALSE:

- Gvpt\_dmatrix
- Gvpt\_dvptnode
- Gvpt\_dpath
- Gvpt\_dlodmult
- Gvpt\_dcnode
- Gvpt\_state\_dtproc
- Gvpt\_state\_efillt
- Gvpt\_state\_edtpc

These variables are maintained using the functions in globs.c. For definitions of the variables, see globs.c.

Called By: cig\_config

Routines Called: vpti\_set\_state\_dcnode  
 vpti\_set\_state\_dlodmult  
 vpti\_set\_state\_dmatrix  
 vpti\_set\_state\_dpath

vpti\_set\_state\_dtproc  
vpti\_set\_state\_dtree\_short  
vpti\_set\_state\_dvptnode  
vpti\_set\_state\_edtpc  
vpti\_set\_state\_efillt

Parameters: none

Returns: none

#### **2.17.8      globs.c (vpti\_\*)**

The globs.c CSU contains functions that manage the global variables that keep track of Viewport Configuration status. These functions can be used to obtain the current value of a variable or to change the variable's setting.

The Viewport Configuration variables that can be managed using these routines are listed in the following table.

Variable	Type	Description
Gvpt_dcnode	BOOLEAN	If TRUE and if debug mode is enabled, status and information messages are output when nodes are configured.
Gvpt_dlodmult	BOOLEAN	<<TBD>>. This variable is used by test routines only.
Gvpt_dmatrix	BOOLEAN	If TRUE and if debug mode is enabled, status and information messages are output when configuration node matrices are manipulated.
Gvpt_dpath	BOOLEAN	If TRUE and if debug mode is enabled, status and information messages are output when graphics path parameters are configured.
Gvpt_dtproc	BOOLEAN	If TRUE and if debug mode is enabled, status and information message are output by cig_config as it processes configuration messages, and by fill_tree as it sets the graphics paths.
Gvpt_dtree_short	BOOLEAN	If TRUE, the information displayed through Gossip for a configuration node is reduced. This variable is used by test routines only.
Gvpt_dvptnode	BOOLEAN	If TRUE and if debug mode is enabled, status and information messages are output when viewport parameters are configured.
Gvpt_edtpc	BOOLEAN	If TRUE and if debug mode is enabled, cig_config exits before calling dtp_compiler.
Gvpt_efillt	BOOLEAN	If TRUE and if debug mode is enabled, cig_config exits before calling fill_tree.
Gvpt_panel_align	BOOLEAN	<<TBD>>
Gvpt_path_entry_index	INT_2	The index of the current entry in the graphics path pointer table.
Gvpt_vpt_entry_index	INT_2	The index of the current entry in the viewport pointer table.
Gvpt_cnode_pointers[ ]	CONFIGURATION_- NODE	Array of pointers to all configuration nodes.
Gvpt_path_pointers[ ]	GRAPHICS_PATH_- PARAMETERS	Array of pointers to all graphics path parameter entries.
Gvpt_vpt_pointers[ ]	VIEWPORT_- PARAMETERS	Array of pointers to all viewport parameter entries.

The routines in globs.c are summarized in the following table. The first column shows the function's name and parameters. The second column describes the function's purpose. The third column identifies the function(s) that call this routine. The functions are listed in alphabetical order, not the order in which they appear in the globs.c file.



globs.c Function	Description	Called By
vpti_change_path_entry_index (value)	Sets Gvpt_path_entry_index to the specified value.	vpt_tree_init
vpti_change_vpt_entry_index (value)	Sets Gvpt_vpt_entry_index to the specified value.	vpt_tree_init
vpti_get_pathptr_cnode (cnode_index)	Returns Gvpt_cnode_pointers [cnode_index]->view_param_ptr->path_param_ptr. Returns 0 if cnode_index is invalid.	none
vpti_get_ptr_cnode(index)	Returns Gvpt_cnode_pointers[index]. Returns 0 if index is invalid.	gos_locate, tst_treetrace, vpt_vpt_set, vpt_cnode_get, vpt_cnode_process, vpt_cnode_groot, vpt_cnode_set_branch, vpt_cnode_linkvpt, vpt_update_mtx, vpt_update_rot, vpt_update_brmask, vpt_update_one_path, vpt_tree_free, vpt_root_init, vptq_dynmtx, vptq_cnout, vptq_cnptrs, vptq_brvals
vpti_get_ptr_path(index)	Returns Gvpt_path_pointers [index]. Returns 0 if index is invalid.	fill_tree, vpt_tree_free, vptq_grptrs, vptq_grout
vpti_get_ptr_vpt(index)	Returns Gvpt_vpt_pointers [index]. Returns 0 if index is invalid.	vpt_tree_free, trav_tree, vptq_vpout, vptq_vpptrs, vptq_vptbrou, vptq_activevpt
vpti_get_vptptr_cnode (cnode_index)	Returns Gvpt_cnode_pointers [cnode_index]->view_param_ptr. Returns 0 if cnode_index is invalid.	vpt_update_all, vpt_update_fov, vpt_update_fov_lod, vpt_update_rez, vpt_update_lodm, vpt_update_near_plane, vpt_update_view_range
vpti_increment_path_entry_index ()	Adds 1 to Gvpt_path_entry_index.	vpt_path_get
vpti_increment_vpt_entry_index()	Adds 1 to Gvpt_vpt_entry_index.	vpt_vpt_get
vpti_set_ptr_cnode(index, ptr)	Sets the specified index in Gvpt_cnode_pointers[] to the specified pointer (ptr).	vpt_cnode_get, vpt_tree_free, vpt_tree_init
vpti_set_ptr_path(index, ptr)	Sets the specified index in Gvpt_path_pointers[] to the specified pointer (ptr).	vpt_tree_free, vpt_tree_init, vpt_path_get
vpti_set_ptr_vpt(index, ptr)	Sets the specified index in Gvpt_vpt_pointers[] to the specified pointer (ptr).	vpt_tree_free, vpt_tree_init, vpt_vpt_get
vpti_set_state_dcnode(value)	Sets Gvpt_dcnode to the specified value.	flagoff, vpt_tree_init
vpti_set_state_dlodmult(value)	Sets Gvpt_dlodmult to the specified value.	flagoff, vpt_tree_init
vpti_set_state_dmatrix(value)	Sets Gvpt_dmatrix to the specified value.	flagoff, vpt_tree_init

vpti_set_state_dpath(value)	Sets Gvpt_dpath to the specified value.	flagoff, vpt_tree_init
vpti_set_state_dtproc(value)	Sets Gvpt_dtproc to the specified value.	flagoff, vpt_tree_init
vpti_set_state_dtree_short(value)	Sets Gvpt_dtree_short to the specified value.	flagoff, vpt_tree_init
vpti_set_state_dvptnode(value)	Sets Gvpt_dvptnode the specified value.	flagoff, vpt_tree_init
vpti_set_state_edtpc(value)	Sets Gvpt_edtpc to the specified value.	flagoff, vpt_tree_init
vpti_set_state_efillt(value)	Sets Gvpt_efillt to the specified value.	flagoff, vpt_tree_init
vpti_set_state_panel_align(value)	Sets Gvpt_panel_align to the specified value.	vpt_tree_init
vpti_state_dcnode()	Returns the value of Gvpt_dcnode.	vpt_cnode_process, vpt_cnode_set_bchild, vpt_cnode_set_stdchild, vpt_cnode_set_branch, vpt_cnode_set_matrix, vpt_update_mtx, vpt_update_rot, vpt_update_brmask, p_configtree_node, tst_edebbug.
vpti_state_dlodmult()	Returns the value of Gvpt_dlodmult.	tst_edebbug
vpti_state_dmatrix()	Returns the value of Gvpt_dmatrix.	concat_mtx, mtx_non_perspective, mtx_perspective, old_mtx_perspective, vpt_path_update, tst_edebbug
vpti_state_dpath()	Returns the value of Gvpt_dpath.	vpt_path_process, vpt_path_query, vpt_path_setup, vpt_path_update, tst_edebbug
vpti_state_dtproc()	Returns the value of Gvpt_dtproc.	cig_config, fill_tree, tst_edebbug
vpti_state_dtree_short()	Returns the value of Gvpt_dtree_short.	tst_edebbug, pr_branch, pr_matrix
vpti_state_dvptnode()	Returns the value of Gvpt_dvptnode.	vpt_vpt_process, vpt_vpt_set, tst_edebbug, p_viewport_state
vpti_state_edtpc()	Returns the value of Gvpt_edtpc.	cig_config, tst_edebbug
vpti_state_efillt()	Returns the value of Gvpt_efillt.	cig_config, tst_edebbug
vpti_state_panel_align()	Returns the value of Gvpt_panel_align.	vpt_path_process, vpt_path_update, tst_edebbug
vpti_togstate_dcnode()	Toggles the state of Gvpt_dcnode.	tst_edebbug
vpti_togstate_dlodmult()	Toggles the state of Gvpt_dlodmult.	tst_edebbug
vpti_togstate_dmatrix()	Toggles the state of Gvpt_dmatrix.	tst_edebbug
vpti_togstate_dpath()	Toggles the state of Gvpt_dpath.	tst_edebbug
vpti_togstate_dtproc()	Toggles the state of Gvpt_dtproc.	tst_edebbug
vpti_togstate_dtree_short()	Toggles the state of Gvpt_dtree_short.	tst_edebbug

<code>vpti_togstate_dvptnode()</code>	Toggles the state of <code>Gvpt_dvptnode</code> .	<code>tst_edebug</code>
<code>vpti_togstate_edtpc()</code>	Toggles the state of <code>Gvpt_edtpc</code> .	<code>tst_edebug</code>
<code>vpti_togstate_efillt()</code>	Toggles the state of <code>Gvpt_efillt</code> .	<code>tst_edebug</code>
<code>vpti_togstate_panel_align()</code>	Toggles the state of <code>Gvpt_panel_align</code> .	<code>tst_edebug</code>
<code>vpti_val_path_entry_index()</code>	Returns the value of <code>Gvpt_path_entry_index</code> .	<code>vpt_path_get</code>
<code>vpti_val_vpt_entry_index()</code>	Returns the value of <code>Gvpt_vpt_entry_index</code> .	<code>vpt_vpt_get</code>

Called By:           see above table

Routines Called:    none

Parameters:          see above table

Returns:             see above table

## 2.17.9      init\_free.c

The functions in the `init_free.c` CSU are used to initialize and remove the viewport configuration tree. These functions are:

- `vpt_root_init`
- `vpt_tree_init`
- `vpt_tree_free`

### 2.17.9.1    vpt\_root\_init

The `vpt_root_init` function allocates memory for the view positions and view flags/branch value arrays. It also sets pointers in the root node to these arrays. This function is called by `vpt_cnode_process` when adding the root node (`node_index = 0`) to the configuration tree.

The function call is `vpt_root_init()`. `vpt_root_init` does the following:

- Gets the pointer to the root node.
- Stores a pointer to the view positions array in the root node's first child pointer (which is otherwise unused).
- Allocates memory for the view positions array in the dynamic section of active area memory.
- Stores a pointer to the branch value array in the root node's `branch_value` pointer.
- Allocates memory for the branch value array in the dynamic section of active area memory.

The function returns SUCCEED if the structures are allocated successfully. It returns FAIL if it could not get a pointer to the root node or allocate memory for one of the arrays.

Called By: vpt\_cnode\_process

Routines Called: aam\_malloc  
printf  
vpti\_get\_ptr\_cnode

Parameters: none

Returns: 1 (SUCCEED)  
0 (FAIL)

#### 2.17.9.2 vpt\_tree\_init

The vpt\_tree\_init function initializes all viewport configuration static variables and static pointer lists. This function is called at system initialization time and when the Simulation Host sends a CIG Control-Stop message to stop the simulation. It can also be invoked through Gossip by selecting the 8 ("configtree init") option from the Gossip main menu, or the I ("Tree Initialize") option from the Viewport Test menu.

The function call is vpt\_tree\_init(). vpt\_tree\_init does the following:

- Sets the following global variables to FALSE:
  - Gvpt\_dmatrix
  - Gvpt\_dvptnode
  - Gvpt\_dpath
  - Gvpt\_dlodmult
  - Gvpt\_dcnode
  - Gvpt\_dtproc
  - Gvpt\_efillt
  - Gvpt\_edtpc
- Sets the following global variables to TRUE:
  - Gvpt\_dtree\_short
  - Gvpt\_panel\_align
- Initializes the viewport parameters entry index and the graphics path parameters entry index to 0.
- Sets the pointer to every configuration node to 0.
- Sets the pointer to every viewport parameter entry to 0
- Sets the pointer to every graphics path entry to 0.

Called By: gossip\_tick  
initialize  
msg\_cig\_ctl  
tst\_tree

Routines Called:     vpti\_change\_path\_entry\_index  
                      vpti\_change\_vpt\_entry\_index  
                      vpti\_set\_ptr\_cnode  
                      vpti\_set\_ptr\_path  
                      vpti\_set\_ptr\_vpt  
                      vpti\_set\_state\_dcnode  
                      vpti\_set\_state\_dlodmult  
                      vpti\_set\_state\_dmatrix  
                      vpti\_set\_state\_dpath  
                      vpti\_set\_state\_dtproc  
                      vpti\_set\_state\_dtree\_short  
                      vpti\_set\_state\_dvptnode  
                      vpti\_set\_state\_edtpc  
                      vpti\_set\_state\_efillt  
                      vpti\_set\_state\_panel\_align

Parameters:           none

Returns:              none

### 2.17.9.3     vpt\_tree\_free

The `vpt_tree_free` function frees the memory allocated for all configuration nodes, viewport parameters, and graphics path parameters. This function is called when the Simulation Host sends a CIG Control-Stop message to stop the simulation. It is also called through Gossip if the user selects the F ("Tree Free") option from the Viewport Test menu.

The function call is `vpt_tree_free()`. `vpt_tree_free` does the following:

- For each configuration node:
  - Gets the pointer to the node.
  - If the root node, frees the view flags/branch value and view positions arrays.
  - Frees the allocated DTP matrix area.
  - Frees the memory allocated for the node.
  - Sets the node's pointer to 0.
- For each viewport parameter entry:
  - Gets the pointer to the entry.
  - Frees the viewing range, near plane, and lod multiplier.
  - Frees the memory allocated for the entry.
  - Sets the entry's pointer to 0.
- For each graphics path parameter entry:
  - Gets the pointer to the entry.
  - Frees the field-of-view vectors, perspective matrix, non-perspective matrix, and resolution.
  - Gets the pointer to the path.
  - Frees the memory allocated for the path.
  - Sets the entry's pointer to 0.

Called By:            msg\_cig\_ctl

tst\_tree

Routines Called:     aam\_free  
                       free  
                       vpti\_get\_ptr\_cnode  
                       vpti\_get\_ptr\_path  
                       vpti\_get\_ptr\_vpt  
                       vpti\_set\_ptr\_cnode  
                       vpti\_set\_ptr\_path  
                       vpti\_set\_ptr\_vpt

Parameters:           none

Returns:               none

### 2.17.10     linkvpt.c (vpt\_cnode\_linkvpt)

The `vpt_cnode_linkvpt` function is responsible for setting a viewport node's pointer to the correct index in the view positions array. This array contains one entry for each simulation vehicle (currently only one). The array is pointed to by the root configuration node.

The function call is `vpt_cnode_linkvpt(vpt_node, ptr_vppos, index)`, where:

*vpt\_node* is a pointer to the viewport parameters node  
*ptr\_vppos* is a pointer to the view positions array  
*index* is the configuration node's index

`vpt_cnode_linkvpt` does the following:

- Calls `vpti_get_ptr_cnode` to get the pointer to the view positions array (stored in the root node's first child pointer slot).
- Calls `vpti_get_ptr_cnode` to get a pointer to the specified configuration node.
- Sets the node's viewport parameters pointer to the specified viewport node.
- Matches the viewport with a hull/world (simulation vehicle) transformation matrix, by finding the vehicle id in the view positions array. Currently, only one simulation vehicle is supported, so there is only one element in the array.

The *status* returned by the function is 1 (SUCCEED) if it is successful, or 0 (FAIL) if the configuration node pointer is NULL or the node index is invalid.

Called By:            vpt\_vpt\_process

Routines Called:     vpti\_get\_ptr\_cnode

Parameters:	VIEWPORT_PARAMETERS	*vpt_node
	R4P3D	**ptr_vppos
	INT_2	index

Returns:                      status

### 2.17.11     **mtx\_concat.c (concat\_mtx)**

The `concat_mtx` function creates or updates the DTP matrix for a specified configuration node, and places the matrix into active area memory. This function is called to generate and load the initial matrix for each matrix node during viewport configuration. It is also called to update dynamic matrices during the simulation if the Simulation Hosts sends any of the following messages: `MSG_ROT2x1_MATRIX`, `MSG_RTS4x3_MATRIX`, `MSG_HPRXYZS_MATRIX`, `MSG_TRANSLATION`, `MSG_SCALE`, `MSG_1ROTATION`, or `MSG_3ROTATIONS`.

The function call is `concat_mtx(config_node, pmtx)`, where:

*config\_node* is a pointer to the configuration node  
*pmtx* is the node's new matrix

`concat_mtx` does the following:

- Verifies that the DTP matrix address is valid.
- If the matrix is defined as dynamic, finds the offset to the current double buffer and finds the location in active area memory for the matrix.
- Unpacks the matrix sent from the Simulation Host.
- For an `RTS4x3` matrix:
  - Calls `mtxcpy` to copy the new matrix to AAM.
- For an `ROT2x1` matrix:
  - Determines which axis the matrix is to be rotated along.
  - Updates the matrix's rotation values in AAM.
- For an `RTS3x3 (HPRXYZS)` matrix:
  - Calls `make4x3` to create the matrix, performing the concatenations in the order specified in the message.
- If this is a dynamic matrix node, calls `mtxcpy` to copy the matrix to the configuration node's entry for use in terrain feedback processing.

If successful, the function returns *err\_code* set to 1 (TRUE). If an invalid DTP matrix address is detected, the function returns no value.

Called By:                      `vpt_cnode_set_matrix`  
                                  `vpt_update_mtx`  
                                  `vpt_update_rot`

Routines Called:              `be_query_buffer_offset`  
                                  `cos`  
                                  `make4x3`  
                                  `mtxcpy`  
                                  `printf`                      (in debug mode only)  
                                  `r4mat_dump`                (in debug mode only)  
                                  `sin`  
                                  `TORAD`

vpti\_state\_dmatrix (in debug mode only)

Parameters: CONFIGURATION\_NODE \*config\_node  
MTXUNION \*pmtx

Returns: err\_code

### 2.17.12 mtx\_dump.c

The functions in the mtx\_dump.c CSU can be used to dump a matrix to stdout. These functions are:

- r4mat\_dump
- r8mat\_dump

#### 2.17.12.1 r4mat\_dump

The r4mat\_dump function dumps a matrix to stdout. This function is called only by test routines or if debug mode is enabled.

The function call is **r4mat\_dump(str, mat)**, where:

*str* is a string to display (on stdout) to describe the matrix

*mat* is a pointer to the location in active area memory that contains the matrix

r4mat\_dump first calls bus\_error to verify that the matrix address is valid. It then outputs the matrix to stdout.

Called By: concat\_mtx (in debug mode only)  
p\_configtree\_node (in debug mode only)  
pr\_matrix  
vpt\_cnode\_set\_matrix (in debug mode only)  
vpt\_path\_update (in debug mode only)  
vptq\_cnout  
vptq\_dynmtx  
vptq\_grout

Routines Called: bus\_error  
printf

Parameters: char \*str  
REAL\_4 mat[4][3]

Returns: none



### 2.17.12.2 r8mat\_dump

The r8mat\_dump function dumps a matrix to stdout.

The function call is **r8mat\_dump(str, mat)**, where:

*str* is a string to display (on stdout) to describe the matrix

*mat* is a pointer to the location in active area memory that contains the matrix

r8mat\_dump first calls bus\_error to verify that the matrix address is valid. It then outputs the matrix to stdout.

This function is not currently used.

Called By: none

Routines Called: bus\_error  
printf

Parameters: char \*str  
REAL\_8 mat[4][3]

Returns: none

### 2.17.13 mtx\_viewspace.c

The functions in the mtx\_viewspace.c CSU generate perspective view matrices for use by the Polygon Processor, and non-perspective view matrices for use by DTP. These functions are:

- mtx\_non\_perspective
- mtx\_perspective
- old\_mtx\_perspective

#### 2.17.13.1 mtx\_non\_perspective

The mtx\_non\_perspective function produces a view matrix without perspective for use by the Poly Processor. This function is called when graphics path parameters are updated.

The function call is **mtx\_non\_perspective(cos\_i, sin\_i, matrix)**, where:

*cos\_i* is the cosine of the graphics path

*sin\_i* is the sine of the graphics path

*matrix* is the destination matrix

Called By: vpt\_path\_update

Routines Called: printf (in debug mode only)  
vpti\_state\_dmatrix (in debug mode only)

Parameters: REAL\_4 cos\_i  
REAL\_4 sin\_i  
T\_MATRIX matrix

Returns: none

### 2.17.13.2 mtx\_perspective

The `mtx_perspective` function produces a view matrix with perspective for use by DTP. When graphics path parameters are updated, this function is called (instead of `old_mtx_perspective`) if the `Gvpt_panel_align` variable is TRUE.

The function call is `mtx_perspective(thx, thy, I, J, zn, paneln, paneli, scale, matrix)`, where:

*thx* is the tangent of one-half the horizontal channel field of view divided by the number of panels

*thy* is the tangent of one-half the vertical channel field of view

*I* is the number of horizontal pixels in the panel

*J* is the number of vertical pixels in the panel

*zn* is the distance to the near plane

*paneln* is the number of panels

*paneli* is the panel number (leftmost is 0)

*scale* is the scale factor used to scale matrices if load module blocking is enabled

*matrix* is the destination matrix

Called By: vpt\_path\_update

Routines Called: printf (in debug mode only)  
vpti\_state\_dmatrix (in debug mode only)

Parameters: float thx  
float thy  
int I  
int J  
float zn  
int paneln  
int paneli  
float scale  
REAL\_4 matrix[4][3]

Returns: none

### 2.17.13.3 old\_mtx\_perspective

The `old_mtx_perspective` function produces a view matrix with perspective for use by DTP. When graphics path parameters are updated, this function is called (instead of `mtx_perspective`) if the `Gvpt_panel_align` variable is FALSE.

The function call is `old_mtx_perspective(itan_i, itan_j, off_x, off_y, cos_i, sin_i, sc, matrix)`, where:

*itan\_i* is the inverse of the tangent of the field-of-view angle i (horizontal)  
*itan\_j* is the inverse of the tangent of the field-of-view angle j (vertical)  
*off\_x* is the horizontal offset  
*off\_y* is the vertical offset  
*cos\_i* is the cosine of the graphics path  
*sin\_i* is the sine of the graphics path  
*sc* is the scale factor used to scale matrices if load module blocking is enabled  
*matrix* is the destination matrix

Called By: vpt\_path\_update

Routines Called: `printf` (in debug mode only)  
`vpti_state_dmatrix` (in debug mode only)

Parameters:	REAL_4	itan_i
	REAL_4	itan_j
	REAL_4	off_x
	REAL_4	off_y
	REAL_4	cos_i
	REAL_4	sin_i
	REAL_4	sc
	T_MATRIX	matrix

Returns: none

### 2.17.14 path.c

The functions in the `path.c` CSU set and update graphics path parameters. These functions are:

- `vpt_path_process`
- `vpt_path_update`

### 2.17.14.1 vpt\_path\_process

The `vpt_path_process` function initializes the fov-related fields in the graphics path parameters and the viewport parameter entries. This function is called when a node with viewport parameters is first defined and when it is updated.

The function call is `vpt_path_process (graphics_path, num_of_paths_i, num_of_paths_j, vpt_rez, vpt_fov, offset)`, where:

*graphics\_path* is a pointer to the graphics path parameters  
*num\_of\_paths\_i* is the number of paths in the i direction  
*num\_of\_paths\_j* is the number of paths in the j direction  
*vpt\_rez* is the viewport's resolution  
*vpt\_fov* is the viewport's field of view  
*offset* is the offset to the current double buffer

`vpt_path_process` processes each graphics path in turn. For each path, it does the following:

- Divides the field of view and resolution by the number of paths.
- Calculates and sets the path's field-of-view vectors.
- If panel alignment is enabled, sets the path's panel align value.
- Sets the path's resolution.
- Calculates the path's center.
- Calls `vpt_path_update` to set the dynamic values in each path.

The function returns SUCCEED if it is successful. It returns FAIL if `vpt_path_update` could not update the path parameters.

Called By:	vpt_update	
Routines Called:	printf	(in debug mode only)
	tan	
	TORAD	
	vpt_path_update	
	vpti_state_dpath	(in debug mode only)
	vpti_state_panel_align	
Parameters:	GRAPHICS_PATH_PARAMETERS	*graphics_path
	INT_2	num_of_paths_i
	INT_2	num_of_paths_j
	I4IJ	vpt_rez
	R4IJ	vpt_fov
	UNS_4	offset
Returns:	1 (SUCCEED)	
	0 (FAIL)	

### 2.17.14.2 vpt\_path\_update

The `vpt_path_update` function updates a graphics path's matrices, field-of-view vectors, and screen constants for DTP. This function is called after the path parameters are initialized.

The function call is `vpt_path_update(graphics_path, offset)`, where:

*graphics\_path* is a pointer to the graphics path  
*offset* is the offset to the current double buffer

`vpt_path_update` does the following:

- Verifies that all of the graphics path parameters are present.
- Calculates the inverse of the tangent of the fov angles.
- Calculates the cosine and sine of the graphics path center.
- Calls `mtx_non_perspective` to build a non-perspective view matrix for DTP.
- Sets the DTP commands for processing of the perspective matrix.
- Calls `mtx_perspective` (if panel alignment is enabled) or `old_mtx_perspective` (if panel alignment is not enabled) to build a perspective view matrix for the Poly Processor.
- Sets the path's field-of-view vectors.
- Sets the path's screen resolution constants.

The function returns SUCCEED if the graphics path is updated successfully. It returns FAIL if any of the graphics path's pointers are NULL.

Called By:	vpt_path_process vpt_update_one_path	
Routines Called:	atan cos mtx_non_perspective mtx_perspective old_mtx_perspective printf r4mat_dump (in debug mode only) sin tan TORAD vpti_state_dmatrix (in debug mode only) vpti_state_dpath (in debug mode only) vpti_state_panel_align	
Parameters:	GRAPHICS_PATH_PARAMETERS UNS_4	*graphics_path offset
Returns:	1 (SUCCEED)	

0 (FAIL)

### 2.17.15 path\_init.c

The functions in the path\_init.c CSU allocate, initialize, and set up graphics path data for configuration nodes that have viewport parameters. These functions are:

- vpt\_path\_get
- vpt\_path\_init
- vpt\_path\_setup

#### 2.17.15.1 vpt\_path\_get

The vpt\_path\_get function allocates memory for the graphics path parameters node and sets the pointers in the path pointer table. This function is called whenever a new viewport parameters node is defined.

The function call is **vpt\_path\_get(num\_paths)**, where *num\_paths* is the number of graphics paths required.

vpt\_path\_get does the following:

- Verifies that the array will hold another pointer.
- Allocates memory for the number of graphics paths specified.
- Verifies that the memory was allocated.
- Adds the entry to the Gvpt\_path\_pointers[] table.
- Increments the path entry index.
- Sets up the graphics path sibling pointers (each points to the next path).

If successful, the functions returns a pointer to the first graphics path structure as *gp*.

Called By: vpt\_path\_setup

Routines Called: calloc  
printf  
vpti\_increment\_path\_entry\_index  
vpti\_set\_ptr\_path  
vpti\_val\_path\_entry\_index

Parameters: INT\_2 num\_paths

Returns: gp

### 2.17.15.2 vpt\_path\_init

The `vpt_path_init` function allocates DTP memory for the perspective matrix, non-perspective matrix, field-of-view vectors, and screen constants. This function is called whenever a new viewport parameters node is defined.

The function call is `vpt_path_init(graphics_path)`, where *graphics\_path* is a pointer to the graphics path parameters.

`vpt_path_init` does the following:

- Allocates memory for the perspective matrix.
- Allocates memory for the non-perspective matrix.
- Allocates memory for the field-of-view vectors.
- Allocates memory for the resolution.
- Verifies that memory was allocated for all of the structures.

Called By: `vpt_path_setup`

Routines Called: `aam_malloc`  
`printf`

Parameters: `GRAPHICS_PATH_PARAMETERS`      `*graphics_path`

Returns: `none`

### 2.17.15.3 vpt\_path\_setup

The `vpt_path_setup` function allocates and initializes graphics path nodes, sets pointers to the viewport node, and sets the path id and sibling pointers. This function is called whenever a node with viewport parameters is added to the configuration tree.

The function call is `vpt_path_setup(viewport, vpt_path_param_ptr, num_of_paths_i, num_of_paths_j, start_path_id)`, where:

*viewport* is a pointer to the viewport parameters  
*vpt\_path\_param\_ptr* is a pointer to the graphics path parameters  
*num\_of\_paths\_i* is the number of paths in the i direction  
*num\_of\_paths\_j* is the number of paths in the j direction  
*start\_path\_id* is the number of the first path in the structure; this varies by type of backend

For each graphics path entry in the structure, `vpt_path_setup` does the following:

- Calls `vpt_path_get` to allocate memory for the graphics paths and get a pointer to the first path structure in the group.
- Sets the viewport pointer to point to the first path.

- Calls `vpt_path_init` to allocate memory for the matrices, fov vectors, etc.
- Sets the graphics path pointer to point to the viewport.
- Sets the graphics path's path id to the start path id.

The function always returns SUCCEED.

Called By:	<code>vpt_vpt_set</code>	
Routines Called:	<code>printf</code> <code>vpt_path_get</code> <code>vpt_path_init</code> <code>vpti_state_dpath</code>	(in debug mode only)  (in debug mode only)
Parameters:	<code>VIEWPORT_PARAMETERS</code> <code>GRAPHICS_PATH_PARAMETERS</code>  <code>INT_2</code> <code>INT_2</code> <code>INT_2</code>	<code>*viewport</code>  <code>**vpt_path_param_ptr</code> <code>num_of_paths_i</code> <code>num_of_paths_j</code> <code>start_path_id</code>
Returns:	<code>1 (SUCCEED)</code>	

### 2.17.16 `path_query.c` (`vpt_path_query`)

The `vpt_path_query` function copies graphics path node information (path id, field-of-view vectors, resolution, screen constants, etc.) to a query structure. This function is used in testing only.

The function call is `vpt_path_query(gp, pi)`, where:

*gp* is a pointer to the graphics path parameters  
*pi* is a pointer to the query structure

`vpt_path_query` zeroes out the query structure before copying the data to it.

Called By:	<code>vptq_grout</code>	
Routines Called:	<code>printf</code> <code>vpti_state_dpath</code>	(in debug mode only) (in debug mode only)
Parameters:	<code>GRAPHICS_PATH_PARAMETERS</code> <code>VPT_PATH_INFO</code>	<code>*gp</code> <code>*pi</code>
Returns:	<code>none</code>	



### 2.17.17 `trav_tree.c`

The `trav_tree` function traverses up each viewport node and sets up a branch array for each viewport. This array can be used to determine which viewports are active at any given time.

The function call is `trav_tree()`.

Called By:	<code>cig_config</code> <code>tst_equery</code>
Routines Called:	<code>printf</code> <code>vpti_get_ptr_vpt</code>
Parameters:	none
Returns:	none

### 2.17.18 `tst_edebug.c`

The `tst_edebug` function reports the state of all Viewport Configuration debug flags and lets the Gossip user toggle them. This function is called if the user selects the 1 ("Change debug switches") option from the Viewport Test menu.

The function call is `tst_edebug()`. `tst_edebug` displays the Current Debug Switches menu showing the current state of each debug flag. (The function calls the `vpti_state_*` functions to determine the current states.) The user can select an option from the menu to toggle the state of the corresponding flag.

The following table identifies the options on the Current Debug Switches menu and shows the steps performed by `tst_edebug` in response to each selection. For definitions of the variables referenced here, see `globals.c`.

Current Debug Switches Menu Option	Processing by tst_edebug
1 Debug Matrix	Calls vpti_togstate_dmatrix to toggle Gvpt_dmatrix.
2 Debug Viewport Node	Calls vpti_togstate_dvptnode to toggle Gvpt_dvptnode.
3 Debug Path	Calls vpti_togstate_dpath to toggle Gvpt_dpath.
4 Debug Config Node	Calls vpti_togstate_dcnode to toggle Gvpt_dcnode.
5 Debug LOD	Calls vpti_togstate_dlodmult to toggle Gvpt_dlodmult.
6 Short tree debug	Calls vpti_togstate_dtree_short to toggle Gvpt_dtree_short.
7 Tree proc debug	Calls vpti_togstate_dtproc to toggle Gvpt_dtproc.
8 Exit at fill tree	Calls vpti_togstate_efillt to toggle Gvpt_efillt.
9 Exit at dtp comp	Calls vpti_togstate_edtpc to toggle Gvpt_edtpc.
a Panel alignment	Calls vpti_togstate_panel_align to toggle Gvpt_panel_align.
x Exit this menu	Exits.

Called By:           tst\_tree

Routines Called:     printf  
                       scanf  
                       vpti\_state\_dcnode  
                       vpti\_state\_dlodmult  
                       vpti\_state\_dmatrix  
                       vpti\_state\_dpath  
                       vpti\_state\_dtproc  
                       vpti\_state\_dtree\_short  
                       vpti\_state\_dvptnode  
                       vpti\_state\_edtpc  
                       vpti\_state\_efillt  
                       vpti\_state\_panel\_align  
                       vpti\_togstate\_dcnode  
                       vpti\_togstate\_dlodmult  
                       vpti\_togstate\_dmatrix  
                       vpti\_togstate\_dpath  
                       vpti\_togstate\_dtproc  
                       vpti\_togstate\_dtree\_short  
                       vpti\_togstate\_dvptnode  
                       vpti\_togstate\_edtpc  
                       vpti\_togstate\_efillt  
                       vpti\_togstate\_panel\_align

Parameters:          none

Returns:             none

### 2.17.19 `tst_equery.c`

The functions in the `tst_equery.c` CSU let the Gossip user query (view) the data in any configuration node, viewport parameters, or graphics path parameters structure. These functions are:

- `tst_equery`
- `vptq_grptrs`
- `vptq_vpptrs`
- `vptq_cnptrs`
- `vptq_brvals`
- `vptq_activept`
- `vptq_vptbrout`
- `vptq_dynmtx`
- `vptq_cnout`
- `vptq_vpout`
- `vptq_grout`

The `tst_equery` function presents a menu from which the user selects the type of data to view. Each `vptq_*` function displays a specific type of data.

#### 2.17.19.1 `tst_equery`

The `tst_equery` function presents a menu to the Gossip user to allow viewing of configuration tree node data, viewport parameters, and graphics path parameters. This function is called if the user selects the 2 ("Query mode") option from the Viewport Test menu.

The function call is `tst_equery`. The following table identifies the options supported by `tst_equery`, and shows the steps performed by `tst_equery` for each user selection.

Query Mode Menu Option	Processing by tst_query
? Display options	Displays this menu.
1 Show confignode pointers table	Calls vptq_cnptrs.
2 Show viewport pointers table	Calls vptq_vpptrs.
3 Show graphics path pointers table	Calls vptq_grptrs.
4 Query confignode given table offset	Calls vptq_cnout.
5 Query viewport given table offset	Calls vptq_vpout.
6 Query graphics path given table offset	Calls vptq_grout.
7 Show branch node info	Calls vptq_brvals.
8 Show matrix node info	Calls vptq_dynmtx.
9 Show viewport branch mask info	Calls vptq_vptbrout.
A Compute active viewports	Calls vptq_activevpt.
B Call trav tree	Calls trav_tree.
X Exit this menu	Exits.

Called By: tst\_tree

Routines Called:

- printf
- scanf
- trav\_tree
- vptq\_activevpt
- vptq\_brvals
- vptq\_cnout
- vptq\_cnptrs
- vptq\_dynmtx
- vptq\_grout
- vptq\_grptrs
- vptq\_vpout
- vptq\_vpptrs
- vptq\_vptbrout

Parameters: none

Returns: none

### 2.17.19.2 vptq\_grptrs

The vptq\_grptrs function displays the graphics path pointers for each node with viewport parameters. This function is called if the Gossip user selects the 3 ("Show graphics path pointers table") option from the Query Mode menu.

The function call is vptq\_grptrs().

Called By:           tst\_equery

Routines Called:     printf  
                      vpti\_get\_ptr\_path

Parameters:          none

Returns:             none

#### 2.17.19.3   vptq\_vpptrs

The vptq\_vpptrs function displays the viewport node pointers for every node with viewport parameters. This function is called if the Gossip user selects the 2 ("Show viewport pointers table") option from the Query Mode menu.

The function call is **vptq\_vpptrs()**.

Called By:           tst\_equery

Routines Called:     printf  
                      vpti\_get\_ptr\_vpt

Parameters:          none

Returns:             none

#### 2.17.19.4   vptq\_cnptrs

The vptq\_cnptrs function displays the configuration node pointers for all nodes in the configuration tree. This function is called if the Gossip user selects the 1 ("Show confignode pointers table") option from the Query Mode menu.

The function call is **vptq\_cnptrs()**.

Called By:           tst\_equery

Routines Called:     printf  
                      vpti\_get\_ptr\_cnode

Parameters:          none

Returns: none

#### 2.17.19.5 vptq\_brvals

The vptq\_brvals function displays information on all branch (conditional) nodes in the configuration tree. The displayed shows each node's node index, branch mask, branch index, current branch value, and description. This function is called if the Gossip user selects the 7 ("Show branch node info") option from the Query Mode menu.

The function call is vptq\_brvals().

Called By: tst\_equery

Routines Called: printf  
scanf  
vpt\_cnode\_query  
vpti\_get\_ptr\_cnode

Parameters: none

Returns: none

#### 2.17.19.6 vptq\_activept

The vptq\_activept function determines the node index and viewport id of each viewport that is currently active, using the array created by trav\_tree. This function is called if the Gossip user selects the A ("Compute active viewports") option from the Query Mode menu.

The function call is vptq\_activept().

Called By: tst\_equery

Routines Called: be\_query\_buffer\_offset  
printf (in debug mode only)  
vpt\_cnode\_qroot  
vpt\_get\_ptr\_vpt

Parameters: none

Returns: none

### 2.17.19.7 `vptq_vptbrout`

The `vptq_vptbrout` function displays branch mask information for each viewport node. The display shows each node's viewport id, node index, and branch value status. This function is called if the Gossip user selects the 9 ("Show viewport branch mask info") option from the Query Mode menu.

The function call is `vptq_vptbrout()`.

Called By:	<code>tst_equery</code>
Routines Called:	<code>printf</code> <code>scanf</code> <code>vpti_get_ptr_vpt</code>
Parameters:	<code>none</code>
Returns:	<code>none</code>

### 2.17.19.8 `vptq_dynmtx`

The `vptq_dynmtx` function displays matrix information for every configuration node defined as a dynamic (changeable) matrix node. The display shows each node's matrix type, description, and current matrix. This function is called if the Gossip user selects the 8 ("Show matrix node info") option from the Query Mode menu.

The function call is `vptq_dynmtx()`.

Called By:	<code>tst_equery</code>
Routines Called:	<code>printf</code> <code>r4mat_dump</code> <code>scanf</code> <code>vpt_cnode_query</code> <code>vpti_get_ptr_cnode</code>
Parameters:	<code>none</code>
Returns:	<code>none</code>

### 2.17.19.9 vptq\_cnout

The vptq\_cnout function displays detailed information describing a specific configuration node. The user is first prompted for the configuration node pointer. The display includes the node's index, parent index, matrix type, branch index, sibling and child pointers, matrix, etc. This function is called if the Gossip user selects the 4 ("Query confignode given table offset") option from the Query Mode menu.

The function call is **vptq\_cnout()**.

Called By:	tst_equery
Routines Called:	printf r4mat_dump scanf vpt_cnode_query vpti_get_ptr_cnode
Parameters:	none
Returns:	none

### 2.17.19.10 vptq\_vpout

The vptq\_vpout function displays detailed information describing a specific viewport node. The user is first prompted for the viewport parameters pointer. The display includes the node index, viewport id, field-of-view parameters, level-of-detail parameters, resolution, near plane, viewing range, etc. This function is called if the Gossip user selects the 5 ("Query viewport given table offset") option from the Query Mode menu.

The function call is **vptq\_vpout()**.

Called By:	tst_equery
Routines Called:	printf scanf vpt_vpt_query vpti_get_ptr_vpt
Parameters:	none
Returns:	none



### 2.17.19.11 vptq\_grout

The vptq\_grout function displays detailed information describing a specific graphics path. The user is first prompted for the graphics path pointer. The display includes the path id, viewport id, panel alignment setting, field-of-view angles, screen resolution, vector data, screen constants, etc. If the node has a sibling, the user is given the option of dumping the sibling's data also. This function is called if the Gossip user selects the 6 ("Query graphics path given table offset") option from the Query Mode menu.

The function call is vptq\_grout().

Called By:           tst\_equery

Routines Called:    printf  
                     r4mat\_dump  
                     scanf  
                     vpt\_path\_query  
                     vpti\_get\_ptr\_path

Parameters:         none

Returns:            none

### 2.17.20     tst\_ereadconfig.c

The functions in the tst\_ereadconfig.c CSU support building a configuration tree from an ASCII file created offline. These functions accommodate testing the viewport configuration process without connection to a Simulation Host.

The functions in tst\_ereadconfig are:

- tst\_ereadconfig
- p\_configtree\_node
- p\_viewport\_state
- p\_overlay\_setup
- setup\_p\_terrain\_feedback
- p\_terrain\_feedback

The messages (commands) that can be included in the ASCII configuration file are defined in the vpi\_msgs.h file. These are not the same messages as those used by the Simulation Host to define a viewport configuration tree.

### 2.17.20.1 `tst_ereadconfig`

The `tst_ereadconfig` function reads and processes an ASCII viewport configuration file created offline. This function is called if the Gossip user selects the "Read configuration file and process" option from the Viewport Test menu.

The function call is `tst_ereadconfig()`. `tst_ereadconfig` does the following:

- Prompts the user to enter the name of the configuration file to be processed.
- Opens the specified file.
- Reads a line from the file into a command buffer.
- Processes the message in the command buffer as follows:
  - `START_CONFIGTREE_NODE` - Calls `p_configtree_node`.
  - `START_VIEWPORT_STATE` - Calls `p_viewport_state`.
  - `START_OVERLAY_SETUP` - Calls `p_overlay_setup`.
  - `START_TERRAIN_FEEDBACK` - Calls `setup_p_terrain_feedback`.
  - `START_TERRAIN_FEEDBACK_POINT_INIT` - Calls `p_terrain_feedback`.
- When an END message is detected, closes the file.

The function exits with a 1 if the specified file cannot be opened.

Called By:	<code>tst_tree</code>
Routines Called:	<code>exit</code> <code>fclose</code> <code>fopen</code> <code>fscanf</code> <code>p_configtree_node</code> <code>p_overlay_setup</code> <code>p_terrain_feedback</code> <code>p_viewport_state</code> <code>printf</code> <code>scanf</code> <code>setup_p_terrain_feedback</code> <code>strcmp</code>
Parameters:	none
Returns:	none

### 2.17.20.2 `p_configtree_node`

The `p_configtree_node` function handles the creation of all configuration nodes specified in the ASCII viewport configuration file. It builds a Simulation Host-type message, then calls the appropriate `vpt_` function to create the configuration node. This function is called if the file specifies `START_CONFIGTREE_NODE`.

The function call is **p\_configtree\_node()**. **p\_configtree\_node** does the following:

- Initializes the MSG\_CREATE\_CONFIGNODE message.
- Reads each parameter in turn from the file into the command buffer.
- Determines the parameter from the command buffer, then reads the corresponding value from the file to build the message.
- Stops reading the file when it detects an END\_CONFIGTREE\_NODE command.
- Calls **vpt\_cnode\_process** to create the node.

Called By: **tst\_ereadconfig**

Routines Called: **fgetc**  
**fscanf**  
**printf**  
**r4mat\_dump** (in debug mode only)  
**strcmp**  
**vpt\_cnode\_process**  
**vpti\_state\_dcnode** (in debug mode only)

Parameters: **none**

Returns: **none**

### 2.17.20.3 **p\_viewport\_state**

The **p\_viewport\_state** function handles the creation of all viewport parameters specified in the ASCII viewport configuration file. It builds a Simulation Host-type message, then calls the appropriate **vpt\_** function to create the viewport parameters. This function is called if the file specifies **START\_VIEWPORT\_STATE**.

The function call is **p\_viewport\_state()**. **p\_viewport\_state** does the following:

- Initializes the MSG\_VIEWPORT\_STATE message.
- Reads each parameter in turn from the file into the command buffer.
- Determines the parameter from the command buffer, then reads the corresponding value from the file to build the message.
- Stops reading the file when it detects an END\_VIEWPORT\_STATE command.
- Calls **vpt\_vpt\_process** to create the viewport parameters.

Called By: **tst\_ereadconfig**

Routines Called: **fscanf**  
**printf**  
**strcmp**  
**vpt\_vpt\_process**  
**vpti\_state\_dvptnode** (in debug mode only)

Parameters: none

Returns: none

#### 2.17.20.4 p\_overlay\_setup

The `p_overlay_setup` function handles the creation of all gun barrel and gunner overlays specified in the ASCII viewport configuration file by building a Simulation Host-type message. This function is called if the file specifies `START_OVERLAY_SETUP`.

The function call is `p_overlay_setup()`. `p_overlay_setup` does the following:

- Initializes the `MSG_OVERLAY_SETUP` message.
- Reads each parameter in turn from the file into the command buffer.
- Determines the parameter from the command buffer, then reads the corresponding value from the file to build the message.
- Stops reading the file when it detects an `END_OVERLAY_SETUP` command.

Called By: `tst_ereadconfig`

Routines Called: `fscanf`  
`printf`  
`strcmp`

Parameters: none

Returns: none

#### 2.17.20.5 setup\_p\_terrain\_feedback

The `setup_p_terrain_feedback` function handles the creation of the terrain feedback header record specified in the ASCII viewport configuration file, by building a Simulation Host-type message. This function is called if the file specifies `START_TERRAIN_FEEDBACK`.

The function call is `setup_p_terrain_feedback()`. The function does the following:

- Initializes a `MSG_TERRAIN_FEEDBACK_SETUP` message.
- Reads the point count from the file and puts it into the message.

Called By: `tst_ereadconfig`

Routines Called: `fscanf`

Parameters: none

Returns: none

#### 2.17.20.6 p\_terrain\_feedback

The p\_terrain\_feedback function handles the creation of all terrain feedback points records specified in the ASCII viewport configuration file, by building a Simulation Host-type message. This function is called if the file specifies START\_TERRAIN\_FEEDBACK\_POINT\_INIT.

The function call is p\_terrain\_feedback(). The function does the following:

- Initializes the MSG\_TERRAIN\_FEEDBACK\_POINT\_INIT message.
- Reads each parameter in turn from the file into the command buffer.
- Determines the parameter from the command buffer, then reads the corresponding value from the file to build the message.
- Stops reading the file when it detects an END\_TERRAIN\_FEEDBACK\_POINT\_INIT command.

Called By: tst\_ereadconfig

Routines Called: fscanf  
printf  
strcmp

Parameters: none

Returns: none

#### 2.17.21 tst\_eupdate.c

The tst\_eupdate function can be used to update configuration node data, viewport parameters, and graphics path parameters for debugging or testing purposes. This function is called if the Gossip user selects the U ("Update config node and viewport information") option from the Viewport Test menu.

The function call is tst\_eupdate(). tst\_eupdate presents a menu that allows the user to select the data to be changed. It then prompts the user for the new values and calls the appropriate vpt\_update\_\* function to make the change.

The following table lists the options supported by tst\_eupdate, and shows the major steps it performs for each selection.

Update Menu Option	Processing by tst_eupdate
0 Update branch mask	Prompts user for configuration node index and branch mask; calls vpt_update_brmask.
1 Update 4x3 matrix	Prompts user for configuration node index, time stamp, and new matrix; calls vpt_update_4x3_matrix.
2 Update 3x3 matrix	Prompts user for configuration node index, time stamp, heading, pitch, roll, x, y, and z; calls vpt_update_3x3_matrix.
3 Update hprxyzs	Prompts user for configuration node index, time stamp, heading, pitch, roll, x, y, and z; calls vpt_update_hprxyzs.
4 Update translation	Prompts user for configuration node index, time stamp, and translation x, y, and z; calls vpt_update_translation.
5 Update scale	Prompts user for configuration node index, time stamp, and scale x, y, and z; calls vpt_update_scale.
6 Update 2x1 heading	Prompts user for configuration node index, time stamp, and heading value; calculates rotation values; calls vpt_update_2x1_heading.
7 Update 2x1 pitch	Prompts user for configuration node index, time stamp, and pitch value; calculates rotation values; calls vpt_update_2x1_pitch.
8 Update 2x1 roll	Prompts user for configuration node index, time stamp, and roll value; calculates rotation values; calls vpt_update_2x1_roll.
9 Update heading	Prompts user for configuration node index, time stamp, and heading value; calls vpt_update_heading.
A Update pitch	Prompts user for configuration node index, time stamp, and pitch value; calls vpt_update_pitch.
B Update roll	Prompts user for configuration node index, time stamp, and roll value; calls vpt_update_roll.
C Update heading, pitch, roll	Prompts user for configuration node index, time stamp, heading, pitch, and roll; calls vpt_update_hpr.
D Update all viewport values	Prompts user for node index, fov angles, resolution, viewing range, near plane, and lod multiplier; calls vpt_update_all.
E Update field of view	Prompts user for node index and fov angles; calls vpt_update_fov.
F Update lod multiplier	Prompts user for node index and lod multiplier; calls vpt_update_lodm.
G Update near plane	Prompts user for node index and near plane; calls vpt_update_near_plane.
H Update resolution	Prompts user for node index and resolution; calls vpt_update_rez.
I Update viewing range	Prompts user for node index and viewing range; calls vpt_update_view_range.
J Update graphics path	Prompts user for node index, path index, path fov, path resolution, and path center in degrees; calls vpt_update_one_path.

X	Exit	Exits.
---	------	--------

Called By:           tst\_tree

Routines Called:     cos  
                      printf  
                      scanf  
                      sin  
                      TORAD  
                      vpt\_update\_2x1\_heading  
                      vpt\_update\_2x1\_pitch  
                      vpt\_update\_2x1\_roll  
                      vpt\_update\_3x3\_matrix  
                      vpt\_update\_4x3\_matrix  
                      vpt\_update\_all  
                      vpt\_update\_brmask  
                      vpt\_update\_fov  
                      vpt\_update\_heading  
                      vpt\_update\_hpr  
                      vpt\_update\_hprxyzs  
                      vpt\_update\_lodm  
                      vpt\_update\_near\_plane  
                      vpt\_update\_one\_path  
                      vpt\_update\_pitch  
                      vpt\_update\_rez  
                      vpt\_update\_roll  
                      vpt\_update\_scale  
                      vpt\_update\_translation  
                      vpt\_update\_view\_range

Parameters:           none

Returns:             none

### 2.17.22     tst\_tree.c

The functions in the `tst_tree.c` CSU are used to present a user interface to the Viewport Configuration test routines. These functions are:

- `tst_tree`
- `mem_check`

#### 2.17.22.1   tst\_tree

The `tst_tree` function provides an interface for the user for debug, update, and test options that can be used to exercise the Viewport Configuration tree. This function is invoked

through Gossip when the user selects the 7 ("configtree menu") option from the Gossip main menu.

The function call is `tst_tree()`. The following table identifies the options displayed and supported by `tst_tree`, and shows the steps it performs for each user selection.

Viewport Test Menu Option	Processing by <code>tst_tree</code>
? Display options	Displays this menu.
0 Read configuration file and process	Calls <code>tst_ereadconfig</code> .
1 Change debug switches	Calls <code>tst_edebug</code> .
2 Query mode	Calls <code>tst_equery</code> .
F Tree Free	Calls <code>vpt_tree_free</code> .
I Tree Initialize	Calls <code>vpt_tree_init</code> .
M Memory alloc test	Calls <code>print_allocp</code> .
T Trace tree information	Calls <code>tst_treetrace</code> .
U Update config node and viewport information	Calls <code>tst_eupdate</code> .
X Exit	Exits.

Called By: `gossip_tick`

Routines Called:

- `print_allocp`
- `printf`
- `scanf`
- `tst_edebug`
- `tst_equery`
- `tst_ereadconfig`
- `tst_eupdate`
- `tst_treetrace`
- `vpt_tree_free`
- `vpt_tree_init`

Parameters: `none`

Returns: `none`

#### 2.17.22.2 `mem_check`

The `mem_check` function calls `malloc` to determine the next available memory address, displays the address, then frees the memory.

The function call is `mem_check()`.

This function is not currently used.



Called By: none

Routines Called: free  
malloc  
printf

Parameters: none

Returns: none

### 2.17.23 tst\_treetrace.c

The functions in the `tst_treetrace.c` CSU are used to trace a node's path through the configuration tree. These functions are:

- `tst_treetrace`
- `pr_branch`
- `pr_matrix`

#### 2.17.23.1 tst\_treetrace

The `tst_treetrace` function, given a node index, displays the node's information and all information about the node's children. This function is called if the Gossip user selects the T ("Tree trace information") option from the Viewport Test menu.

The function call is `tst_treetrace()`. The function does the following:

- Prompts the user for the starting node index.
- Calls `vpti_get_ptr_cnode` to get a pointer to the starting node.
- Calls `vpt_cnode_query` to get information on the starting node.
- If the node is a branch (conditional) node, calls `pr_branch` to display the information. If the node is a matrix node, calls `pr_matrix` to display the information.
- If the node has children, repeats the above process for each child node.
- Prompts the user for another starting node; repeats until the user enters 999.

Called By: `tst_tree`

Routines Called: `pr_branch`  
`pr_matrix`  
`printf`  
`scanf`  
`vpt_cnode_query`  
`vpti_get_ptr_cnode`

Parameters: none

Returns: none

### 2.17.23.2 pr\_branch

The `pr_branch` function displays information on branch (conditional) nodes. This function is called by `tst_treetrace` if the node to be displayed is defined as a branch node.

The function call is `pr_branch(info)`, where *info* is a pointer to the configuration node information structure.

`pr_branch` outputs the node's node index, branch index, branch mask, and description. If the `Gvpt_dtree_short` variable is set to `FALSE`, it also outputs the node's parent index, true child flag, and vehicle id.

Called By: `tst_treetrace`

Routines Called: `printf`  
`vpti_state_dtree_short`

Parameters: `VPT_CNODE_INFO` \*info

Returns: none

### 2.17.23.3 pr\_matrix

The `pr_matrix` function displays information on matrix nodes. This function is called by `tst_treetrace` if the node to be displayed is defined as a matrix node.

The function call is `pr_matrix(info)`, where *info* is a pointer to the configuration node information structure.

`pr_matrix` outputs the node's node index, matrix type, static matrix flag, branch mask, and description. If the node has viewport parameters attached, `pr_matrix` outputs a statement to that effect.

If the `Gvpt_dtree_short` variable is set to `FALSE`, `pr_matrix` also outputs the node's parent index, true child flag, vehicle id, and, if the matrix type is `RTS4x3`, the node's matrix.

Called By: `tst_treetrace`

Routines Called: `printf`  
`r4mat_dump`

vpti\_state\_dtree\_short

Parameters: VPT\_CNODE\_INFO \*info

Returns: none

**2.17.24 u\_brmask.c (vpt\_update\_brmask)**

The `vpt_update_brmask` function updates (changes) the branch mask of a specified configuration node. This function is called if the Gossip user selects the 0 ("Update branch mask") option from the Update menu.

The function call is `vpt_update_brmask(node_index, br_mask)`, where:

*node\_index* is the unique index used to reference this node  
*br\_mask* is the new branch mask

`vpt_update_brmask` does the following:

- Gets a pointer to the node.
- Makes sure this is *not* the root node (`node_index = 0`).
- Updates the node's branch mask.

The function returns `SUCCEED` if it is successful. It returns `FAIL` if the specified node is the root node, or if it could not get a pointer to the node.

Called By: tst\_eupdate

Routines Called: `printf`  
`vpti_get_ptr_cnode`  
`vpti_state_dcnode` (in debug mode only)

Parameters: `INT_2` `node_index`  
`UNS_4` `br_mask`

Returns: 1 (`SUCCEED`)  
0 (`FAIL`)

**2.17.25 u\_path.c (vpt\_update\_one\_path)**

The `vpt_update_one_path` function updates a single graphics path's parameters (field-of-view angles, resolution, and path center). This function is called if the Gossip user selects the J ("Update graphics path") option from the Update menu.

The function call is `vpt_update_one_path(node_index, path_id, fov, rez, center)`, where:

*node\_index* is the unique index used to reference this node  
*path\_id* identifies the graphics path  
*fov* is the path's field-of-view  
*rez* is the path's resolution  
*center* is the center of the path in degrees

*vpt\_update\_one\_path* does the following:

- Gets a pointer to the specified node.
- Verifies that the node has viewport parameters attached to it.
- Sets the new values.
- Calls *vpt\_path\_update* to update the path with the new parameters.

The function returns SUCCEED if it is successful. It returns FAIL if it cannot get a pointer to the node, the node does not have viewport parameters, or *vpt\_path\_update* cannot update the graphics path.

Called By:            *tst\_eupdate*

Routines Called:    *be\_query\_buffer\_offset*  
                       *vpt\_path\_update*  
                       *vpti\_get\_ptr\_cnode*

Parameters:	INT_2	<i>node_index</i>
	INT_2	<i>path_id</i>
	R4IJ	<i>fov</i>
	I4IJ	<i>rez</i>
	REAL_4	<i>center</i>

Returns:            1 (SUCCEED)  
                       0 (FAIL)

## 2.17.26      *u\_rotations.c*

The functions in the *u\_rotations.c* CSU update part or all of the rotation matrix for a given configuration node. These functions are:

- *vpt\_update\_2x1\_heading*
- *vpt\_update\_2x1\_pitch*
- *vpt\_update\_2x1\_roll*
- *vpt\_update\_heading*
- *vpt\_update\_pitch*
- *vpt\_update\_roll*
- *vpt\_update\_hpr*

### 2.17.26.1 vpt\_update\_2x1\_heading

The `vpt_update_2x1_heading` function is used to update a matrix node's heading. This function is called when the Simulation Host sends a `MSG_ROT2x1_MATRIX` message to rotate an `RTS4x3` matrix around axis 0. It is also invoked through Gossip if the user selects the 6 ("Update 2x1 heading") option from the Update menu.

The function call is `vpt_update_2x1_heading(node_index, time_stamp, cosrotation, sinrotation)`, where:

*node\_index* is the unique index used to reference this node  
*time\_stamp* is the time stamped in the Simulation Host message  
*cosrotation* is the cosine of the angle of rotation  
*sinrotation* is the sine of the angle of rotation

`vpt_update_2x1_heading` calls `vpt_update_rot` to perform the indicated rotation. It then returns the *status* returned from `vpt_update_rot`: SUCCEED or FAIL.

Called By:            `msg_rot2x1_matrix`  
                      `tst_eupdate`

Routines Called:    `printf`  
                      `vpt_update_rot`

Parameters:	<code>INT_2</code>	<code>node_index</code>
	<code>INT_2</code>	<code>time_stamp</code>
	<code>REAL_4</code>	<code>cosrotation</code>
	<code>REAL_4</code>	<code>sinrotation</code>

Returns:            `status`

### 2.17.26.2 vpt\_update\_2x1\_pitch

The `vpt_update_2x1_pitch` function is used to update a matrix node's pitch. This function is called when the Simulation Host sends a `MSG_ROT2x1_MATRIX` message to rotate an `RTS4x3` matrix around axis 1. It is also invoked through Gossip if the user selects the 7 ("Update 2x1 pitch") option from the Update menu.

The function call is `vpt_update_2x1_pitch(node_index, time_stamp, cosrotation, sinrotation)`, where:

*node\_index* is the unique index used to reference this node  
*time\_stamp* is the time stamped in the Simulation Host message  
*cosrotation* is the cosine of the angle of rotation  
*sinrotation* is the sine of the angle of rotation

`vpt_update_2x1_pitch` calls `vpt_update_rot` to perform the indicated rotation. It then returns the *status* returned from `vpt_update_rot`: SUCCEED or FAIL.

Called By:	<code>msg_rot2x1_matrix</code> <code>tst_eupdate</code>	
Routines Called:	<code>printf</code> <code>vpt_update_rot</code>	
Parameters:	<code>INT_2</code> <code>INT_2</code> <code>REAL_4</code> <code>REAL_4</code>	<code>node_index</code> <code>time_stamp</code> <code>cosrotation</code> <code>sinrotation</code>
Returns:	<code>status</code>	

### 2.17.26.3 `vpt_update_2x1_roll`

The `vpt_update_2x1_roll` function is used to update a matrix node's roll. This function is called when the Simulation Host sends a `MSG_ROT2x1_MATRIX` message to rotate an `RTS4x3` matrix around axis 2. It is also invoked through Gossip if the user selects the 8 ("Update 2x1 roll") option from the Update menu.

The function call is `vpt_update_2x1_roll(node_index, time_stamp, cosrotation, sinrotation)`, where:

*node\_index* is the unique index used to reference this node  
*time\_stamp* is the time stamped in the Simulation Host message  
*cosrotation* is the cosine of the angle of rotation  
*sinrotation* is the sine of the angle of rotation

`vpt_update_2x1_roll` calls `vpt_update_rot` to perform the indicated rotation. It then returns the *status* returned from `vpt_update_rot`: SUCCEED or FAIL.

Called By:	<code>msg_rot2x1_matrix</code> <code>tst_eupdate</code>	
Routines Called:	<code>printf</code> <code>vpt_update_rot</code>	
Parameters:	<code>INT_2</code> <code>INT_2</code> <code>REAL_4</code> <code>REAL_4</code>	<code>node_index</code> <code>time_stamp</code> <code>cosrotation</code> <code>sinrotation</code>

Returns: status

#### 2.17.26.4 vpt\_update\_heading

The `vpt_update_heading` function is used to update a matrix node's heading. This function is called when the Simulation Host sends a MSG\_1ROTATION message to rotate an HPRXYZS matrix around axis 0 (heading). It is also invoked through Gossip if the user selects the 9 ("Update heading") option from the Update menu.

The function call is `vpt_update_heading(node_index, time_stamp, rotation)`, where:

*node\_index* is the unique index used to reference this node  
*time\_stamp* is the time stamped in the Simulation Host message  
*rotation* is the angle of rotation in degrees

`vpt_update_heading` calls `vpt_update_rot` to perform the indicated rotation. It then returns the *status* returned from `vpt_update_rot`: SUCCEED or FAIL.

Called By: msg\_1rotation  
tst\_eupdate

Routines Called: printf  
vpt\_update\_rot

Parameters: INT\_2 node\_index  
INT\_2 time\_stamp  
REAL\_4 rotation

Returns: status

#### 2.17.26.5 vpt\_update\_pitch

The `vpt_update_pitch` function is used to update a matrix node's pitch. This function is called when the Simulation Host sends a MSG\_1ROTATION message to rotate an HPRXYZS matrix around axis 1 (pitch). It is also invoked through Gossip if the user selects the A ("Update pitch") option from the Update menu.

The function call is `vpt_update_pitch(node_index, time_stamp, rotation)`, where:

*node\_index* is the unique index used to reference this node  
*time\_stamp* is the time stamped in the Simulation Host message  
*rotation* is the angle of rotation in degrees

`vpt_update_pitch` calls `vpt_update_rot` to perform the indicated rotation. It then returns the *status* returned from `vpt_update_rot`: SUCCEED or FAIL.

Called By:	msg_1rotation tst_eupdate	
Routines Called:	printf vpt_update_rot	
Parameters:	INT_2 INT_2 REAL_4	node_index time_stamp rotation
Returns:	status	

#### 2.17.26.6 vpt\_update\_roll

The `vpt_update_roll` function is used to update a matrix node's roll. This function is called when the Simulation Host sends a MSG\_1ROTATION message to rotate an HPRXYZS matrix around axis 2 (roll). It is also invoked through Gossip if the user selects the B ("Update roll") option from the Update menu.

The function call is `vpt_update_roll(node_index, time_stamp, rotation)`, where:

*node\_index* is the unique index used to reference this node  
*time\_stamp* is the time stamped in the Simulation Host message  
*rotation* is the angle of rotation in degrees

`vpt_update_roll` calls `vpt_update_rot` to perform the indicated rotation. It then returns the *status* returned from `vpt_update_rot`: SUCCEED or FAIL.

Called By:	msg_1rotation tst_eupdate	
Routines Called:	printf vpt_update_rot	
Parameters:	INT_2 INT_2 REAL_4	node_index time_stamp rotation
Returns:	status	

#### 2.17.26.7 vpt\_update\_hpr

The `vpt_update_hpr` function is used to update a matrix node's heading, pitch, and roll. This function is called when the Simulation Host sends a MSG\_3ROTATIONS message



for an HPRXYZS matrix. It is also invoked through Gossip if the user selects the C ("Update heading, pitch, roll") option from the Update menu.

The function call is `vpt_update_hpr(node_index, time_stamp, h, p, r)`, where:

*node\_index* is the unique index used to reference this node  
*time\_stamp* is the time stamped in the Simulation Host message  
*h* is the new heading in degrees  
*p* is the new pitch in degrees  
*r* is the new roll in degrees

`vpt_update_hpr` calls `vpt_update_rot` to perform the indicated rotations. It then returns the *status* returned from `vpt_update_rot`: SUCCEED or FAIL.

Called By:            `msg_3rotations`  
                       `tst_eupdate`

Routines Called:    `printf`  
                       `vpt_update_rot`

Parameters:	<code>INT_2</code>	<code>node_index</code>
	<code>INT_2</code>	<code>time_stamp</code>
	<code>REAL_4</code>	<code>h</code>
	<code>REAL_4</code>	<code>p</code>
	<code>REAL_4</code>	<code>r</code>

Returns:            `status`

### 2.17.27    `u_viewport.c`

The functions in the `u_viewport.c` CSU are used to update a configuration node's viewport parameters. These functions are:

- `vpt_update_fov`
- `vpt_update_fov_lod`
- `vpt_update_lodm`
- `vpt_update_near_plane`
- `vpt_update_rez`
- `vpt_update_view_range`
- `vpt_update_all`

Most of these functions are currently used only for testing (via Gossip).

#### 2.17.27.1    `vpt_update_fov`

The `vpt_update_fov` function is used to update a viewport node's field-of-view. This function is invoked through Gossip if the user selects the E ("Update field of view") option from the Update menu.

The function call is **vpt\_update\_fov(index, fov)**, where:

*index* is the viewport node's index  
*fov* is the new field-of-view

**vpt\_update\_fov** gets a pointer to the viewport parameters, then calls **vpt\_update** to perform the update. The function returns the *status* (SUCCEED or FAIL) returned from **vpt\_update**. It returns FAIL if it could not get a pointer to the viewport parameters.

Called By:            **tst\_eupdate**

Routines Called:    **be\_query\_buffer\_offset**  
                      **vpt\_update**  
                      **vpti\_get\_vptptr\_cnode**

Parameters:            INT\_2                            index  
                      R4IJ                                fov

Returns:               status  
                         0 (FAIL)

#### 2.17.27.2 **vpt\_update\_fov\_lod**

The **vpt\_update\_fov\_lod** function updates a viewport node's field-of-view and level-of-detail multiplier. This function is called when the Simulation Host sends a **MSG\_VIEW\_MAGNIFICATION** message.

The function call is **vpt\_update\_fov\_lod(index, fov, lod)**, where:

*index* is the viewport node's index  
*fov* is the new field-of-view value  
*lod* is the new level-of-detail value

**vpt\_update\_fov\_lod** gets a pointer to the viewport parameters, then calls **vpt\_update** to perform the update. The function returns the *status* (SUCCEED or FAIL) returned from **vpt\_update**. It returns FAIL if it could not get a pointer to the viewport parameters.

Called By:            **msg\_view\_magnification**

Routines Called:    **be\_query\_buffer\_offset**  
                      **vpt\_update**  
                      **vpti\_get\_vptptr\_cnode**

Parameters:            INT\_2                            index  
                      R4IJ                                fov

REAL\_4

lod

Returns:           status  
                  0 (FAIL)

### 2.17.27.3   vpt\_update\_lodm

The `vpt_update_lodm` function is used to update a viewport node's level-of-detail multiplier. This function is invoked through Gossip if the user selects the F ("Update lod multiplier") option from the Update menu.

The function call is `vpt_update_lodm(index, lod_multiplier)`, where:

*index* is the viewport node's index  
*lod\_multiplier* is the new level-of-detail

`vpt_update_lodm` gets a pointer to the viewport parameters, then calls `vpt_update` to perform the update. The function returns the *status* (SUCCEED or FAIL) returned from `vpt_update`. It returns FAIL if it could not get a pointer to the viewport parameters.

Called By:           tst\_eupdate

Routines Called:    be\_query\_buffer\_offset  
                      vpt\_update  
                      vpti\_get\_vptptr\_cnode

Parameters:          INT\_2                           index  
                      REAL\_4                       lod\_multiplier

Returns:            status  
                      0 (FAIL)

### 2.17.27.4   vpt\_update\_near\_plane

The `vpt_update_near_plane` function is used to update a viewport node's near plane (the closest point that can be seen from the viewport). This function is invoked through Gossip if the user selects the G ("Update near plane") option from the Update menu.

The function call is `vpt_update_near_plane(index, near_plane)`, where:

*index* is the viewport node's index  
*near\_plane* is the near plane value

`vpt_update_near_plane` gets a pointer to the viewport parameters, then calls `vpt_update` to perform the update. The function returns the *status* (SUCCEED or FAIL) returned from `vpt_update`. It returns FAIL if it could not get a pointer to the viewport parameters.

Called By:           tst\_eupdate

Routines Called:    be\_query\_buffer\_offset  
                     vpt\_update  
                     vpti\_get\_vptptr\_cnode

Parameters:         INT\_2                               index  
                     REAL\_4                             near\_plane

Returns:            status  
                     0 (FAIL)

#### 2.17.27.5 vpt\_update\_rez

The `vpt_update_rez` function is used to update a viewport node's screen resolution. This function is invoked through Gossip if the user selects the H ("Update resolution") option from the Update menu.

The function call is `vpt_update_rez(index, rez)`, where:

*index* is the viewport node's index  
*rez* is the new screen resolution

`vpt_update_rez` gets a pointer to the viewport parameters, then calls `vpt_update` to perform the update. The function returns the *status* (SUCCEED or FAIL) returned from `vpt_update`. It returns FAIL if it could not get a pointer to the viewport parameters.

Called By:           tst\_eupdate

Routines Called:    be\_query\_buffer\_offset  
                     vpt\_update  
                     vpti\_get\_vptptr\_cnode

Parameters:         INT\_2                               index  
                     I4IJ                               rez

Returns:            status  
                     0 (FAIL)

#### 2.17.27.6 vpt\_update\_view\_range

The `vpt_update_view_range` function is used to update a viewport node's viewing range (the distance that can be seen from the viewport). This function is invoked through Gossip if the user selects the I ("Update viewing range") option from the Update menu.



Parameters:	INT_2 R4IJ I4IJ REAL_4 REAL_4 REAL_4	index fov rez viewing_range near_plane lod_multiplier
Returns:	status 0 (FAIL)	

### 2.17.28 u\_xfrm.c

The functions in the u\_xfrm.c CSU process changes to the transformation matrices in the configuration tree. These functions are:

- vpt\_update\_4x3\_matrix
- vpt\_update\_3x3\_matrix
- vpt\_update\_hprxyzs
- vpt\_update\_scale
- vpt\_update\_translation

#### 2.17.28.1 vpt\_update\_4x3\_matrix

The vpt\_update\_4x3\_matrix function is used to replace the matrix of a configuration node defined with a matrix type of RTS4x3\_TYPE. This function is called when the Simulation Host sends a MSG\_RTS4x3\_MATRIX message. It is also invoked through Gossip if the user selects the 1 ("Update 4x3 matrix") option from the Update menu.

The function call is **vpt\_update\_4x3\_matrix(node\_index, time\_stamp, matrix)**, where:

*node\_index* is the unique index used to reference this node  
*time\_stamp* is the time stamped in the Simulation Host message  
*matrix* is the new transformation matrix

vpt\_update\_4x3\_matrix calls vpt\_update\_mtx to perform the actual update, then returns the *status* (SUCCEED or FAIL) returned from vpt\_update\_mtx.

Called By:	msg_rts4x3_matrix tst_eupdate	
Routines Called:	printf vpt_update_mtx	
Parameters:	INT_2 INT_2	node_index time_stamp

MTXUNION

\*matrix

Returns: status

### 2.17.28.2 vpt\_update\_3x3\_matrix

The `vpt_update_3x3_matrix` function is used to replace the matrix of a configuration node defined with a matrix type of `RTS3x3_TYPE`. This function is invoked through Gossip if the user selects the 2 ("Update 3x3 matrix") option from the Update menu.

The function call is `vpt_update_3x3_matrix(node_index, time_stamp, matrix)`, where:

*node\_index* is the unique index used to reference this node

*time\_stamp* is the time stamped in the Simulation Host message

*matrix* is the new transformation matrix

`vpt_update_3x3_matrix` calls `vpt_update_mtx` to perform the actual update, then returns the *status* (SUCCEED or FAIL) returned from `vpt_update_mtx`.

Called By: tst\_eupdate

Routines Called: printf  
vpt\_update\_mtxParameters: INT\_2 node\_index  
INT\_2 time\_stamp  
MTXUNION \*matrix

Returns: status

### 2.17.28.3 vpt\_update\_hprxyzs

The `vpt_update_hprxyzs` function is used to replace the matrix of a configuration node defined with a matrix type of `RTS3x3_TYPE`. This function is called when the Simulation Host sends a `MSG_HPRXYZS` message. It is also invoked through Gossip if the user selects the 3 ("Update hprxyzs") option from the Update menu.

The function call is `vpt_update_hprxyzs(node_index, time_stamp, matrix)`, where:

*node\_index* is the unique index used to reference this node

*time\_stamp* is the time stamped in the Simulation Host message

*matrix* is the new transformation matrix

`vpt_update_hprxyzs` calls `vpt_update_mtx` to perform the actual update, then returns the *status* (SUCCEED or FAIL) returned from `vpt_update_mtx`.

Called By:	msg_hprxyzs_matrix tst_eupdate	
Routines Called:	printf vpt_update_mtx	
Parameters:	INT_2 INT_2 MTXUNION	node_index time_stamp *matrix
Returns:	status	

#### 2.17.28.4 vpt\_update\_scale

The `vpt_update_scale` function is used to change the scale value of a configuration node defined with a matrix type of `HPRXYZS_TYPE`. This function is called when the Simulation Host sends a `MSG_SCALE` message. It is also invoked through Gossip if the user selects the 5 ("Update scale") option from the Update menu.

The function call is `vpt_update_scale(node_index, time_stamp, x, y, z)`, where:

*node\_index* is the unique index used to reference this node  
*time\_stamp* is the time stamped in the Simulation Host message  
*x* is the new x scale factor  
*y* is the new y scale factor  
*z* is the new z scale factor

`vpt_update_scale` calls `vpt_update_mtx` to perform the actual update, then returns the *status* (SUCCEED or FAIL) returned from `vpt_update_mtx`.

*Note: The MSG\_SCALE message is reserved for future expansion.*

Called By:	msg_scale tst_eupdate	
Routines Called:	printf vpt_update_mtx	
Parameters:	INT_2 INT_2 REAL_4 REAL_4 REAL_4	node_index time_stamp x y z



Returns: status

### 2.17.28.5 vpt\_update\_translation

The `vpt_update_translation` function is used to change the translation value of a configuration node defined with a matrix type of `HPRXYZS_TYPE`. This function is called when the Simulation Host sends a `MSG_TRANSLATION` message. It is also invoked through Gossip if the user selects the 4 ("Update translation") option from the Update menu.

The function call is `vpt_update_translation(node_index, time_stamp, x, y, z)`, where:

*node\_index* is the unique index used to reference this node  
*time\_stamp* is the time stamped in the Simulation Host message  
*x* is the new x translation value  
*y* is the new y translation value  
*z* is the new z translation value

`vpt_update_translation` calls `vpt_update_mtx` to perform the actual update, then returns the *status* (SUCCEED or FAIL) returned from `vpt_update_mtx`.

Called By: msg\_translation  
 tst\_eupdate

Routines Called: printf  
 vpt\_update\_mtx

Parameters:	INT_2	node_index
	INT_2	time_stamp
	REAL_4	x
	REAL_4	y
	REAL_4	z

Returns: status

### 2.17.29 update\_mtx.c (vpt\_update\_mtx)

The `vpt_update_mtx` function replaces or modifies the transformation matrices in the configuration tree. This function is called by the functions in the `u_xfrm.c` CSU, which in turn are called when the Simulation Host sends messages to update matrices, and when the Gossip user selects an option to change a matrix.

The function call is `vpt_update_mtx(node_index, time_stamp, matrix_type, matrix)`, where:

*node\_index* is the unique index used to reference this node  
*time\_stamp* is the time stamped in the Simulation Host message

*matrix\_type* is 0 (HPRXYZS matrix), 1 (RTS4x3 matrix), or 2 (ROT2x1 matrix)  
*matrix* is a pointer to the new matrix

*vpt\_update\_mtx* does the following:

- Gets a pointer to the node.
- Verifies that this is not the root node; returns an error and terminates if it is.
- If the complete matrix is being changed, calls *mtxcpy* to copy the new matrix to the configuration node.
- If only the scale or translation is being changed, changes the values in the matrix in the configuration node.
- Calls *concat\_mtx* to update the copy of the matrix used by DTP.
- If this is a hull/world node (i.e., a child of the root node), calls *process\_vppos* to update the simulated vehicle's current position.

The function returns SUCCEED if it is able to update the matrix. It returns FAIL if it cannot get a pointer to the node or the specified node is the root node.

Called By:	<i>vpt_update_3x3_matrix</i> <i>vpt_update_4x3_matrix</i> <i>vpt_update_hprxyzs</i> <i>vpt_update_scale</i> <i>vpt_update_translation</i>	
Routines Called:	<i>concat_mtx</i> <i>mtxcpy</i> <i>printf</i> <i>process_vppos</i> <i>vpti_get_ptr_cnode</i> <i>vpti_state_dcnode</i>	(in debug mode only)
Parameters:	INT_2 INT_2 UNS_1	node_index time_stamp matrix_type
Returns:	1 (SUCCEED) 0 (FAIL)	

### 2.17.30 update\_rot.c (*vpt\_update\_rot*)

The *vpt\_update\_rot* function updates the rotation values in the transformation matrices in the configuration tree. This function is called by the functions in the *u\_rotations.c* CSU, which in turn are called when the Simulation Host sends messages to rotate matrices, or the Gossip user selects an option to rotate a matrix.

The function call is *vpt\_update\_rot*(*node\_index*, *time\_stamp*, *type*, *rotation\_axis*, *rotation*), where:

*node\_index* is the unique index used to reference this node

*time\_stamp* is the time stamped in the Simulation Host message  
*type* is the type of rotation: ROT2x1\_TYPE, ROT1\_TYPE (for one rotation), or  
 ROT3\_TYPE (for three rotations)  
*rotation\_axis* is the axis to be rotated: 0 (heading), 1 (pitch), or 2 (roll)  
*rotation* is the angle of rotation

vpt\_update\_rot does the following:

- Gets a pointer to the node.
- Verifies that this is not the root node; returns an error and terminates if it is.
- Updates the rotation value(s) in the node's matrix, based on the node's matrix type and the rotation type specified in the call.
- Calls concat\_mtx to update the copy of the matrix used by DTP.
- If this is a hull/world node (i.e., a child of the root node), calls process\_vpops to update the simulated vehicle's current position.

The function returns SUCCEED if it is able to update the matrix. It returns FAIL if it cannot get a pointer to the node or the specified node is the root node.

Called By:	vpt_update_2x1_heading vpt_update_2x1_pitch vpt_update_2x1_roll vpt_update_heading vpt_update_hpr vpt_update_pitch vpt_update_roll	
Routines Called:	concat_mtx printf process_vpops vpti_get_ptr_cnode vpti_state_dcnode	(in debug mode only)
Parameters:	INT_2 INT_2 UNS_1 UNS_1 REAL_4	node_index time_stamp type rotation_axis *rotation
Returns:	1 (SUCCEED) 0 (FAIL)	

### 2.17.31 vpt\_get.c (vpt\_vpt\_get)

The vpt\_vpt\_get function allocates memory for a new viewport parameter node and returns a pointer to it. This function is called whenever a configuration node with viewport parameters attached to it is defined.

The function call is vpt\_vpt\_get(). The function does the following:

- Verifies that the viewport parameters array will hold another pointer.
- Allocates memory for the new viewport parameters.
- Verifies that the memory was allocated.
- Places the new entry in the viewport parameters pointers table (Gvpt\_vpt\_pointers).
- Increments the Gvpt\_vpt\_entry\_index.

If successful, the function returns a pointer to the viewport parameters entry as *viewpt*.

Called By: vpt\_vpt\_process

Routines Called: calloc  
printf  
vpti\_increment\_vpt\_entry\_index  
vpti\_set\_ptr\_vpt  
vpti\_val\_vpt\_entry\_index

Parameters: none

Returns: viewpt

### 2.17.32 vpt\_process.c (vpt\_vpt\_process)

The vpt\_vpt\_process function creates and initializes viewport parameters entries in the viewport configuration tree. This function is called whenever a node with viewport parameters attached to it is defined.

The function call is **vpt\_vpt\_process(node\_index, viewport\_id, database\_id, fov, resolution, viewing\_range, near\_plane, lod\_multiplier, aspect\_ratio, n\_vpts)**, where:

*node\_index* is the unique index used to reference this node  
*viewport\_id* is the identifier assigned to this viewport  
*database\_id* is the database identifier (for future use)  
*fov* is the viewport's field-of-view  
*resolution* is the viewport's screen resolution  
*viewing\_range* is the distance that can be viewed from the viewport  
*near\_plane* is the closest point that can be seen from the viewport  
*lod\_multiplier* is the viewport's level-of-detail multiplier  
*aspect\_ratio* is the ratio of the sides (width:height) of the viewport (for future use)  
*n\_vpts* is the number of viewports

vpt\_vpt\_process does the following:

- Calls vpt\_vpt\_get to get a pointer for the new viewport node.
- Calls vpt\_cnode\_linkvpt to set the node's pointer to the correct index in the view positions array.
- Calls be\_query\_num\_paths to determine how many graphics paths are needed.
- Sets the offset to 0 (for double buffer 0).

- Calls `be_query_lm_per_lmb_side` to get the number of load modules per load module block side. This value is used to scale the images in the viewport for the larger viewing range if load module blocking is enabled.
- Set the number of viewports.
- Calls `vpt_vpt_set` to place the viewport parameter data in the configuration tree.

The function returns **SUCCEED** if the viewport parameters are initialized successfully. It returns **FAIL** if the pointer to the `vppos` array could not be set, graphics paths could not be allocated, or viewport setup failed.

Called By:            `cig_config`  
                      `p_viewport_state`

Routines Called:    `be_query_lm_per_lmb_side`  
                      `be_query_num_paths`  
                      `printf`  
                      `vpt_cnode_linkvpt`  
                      `vpt_vpt_get`  
                      `vpt_vpt_set`  
                      `vpti_state_dvptnode`            (in debug mode only)

Parameters:	<code>INT_2</code>	<code>node_index</code>
	<code>INT_2</code>	<code>viewport_id</code>
	<code>UNS_1</code>	<code>database_id</code>
	<code>R4IJ</code>	<code>fov</code>
	<code>I4IJ</code>	<code>resolution</code>
	<code>REAL_4</code>	<code>viewing_range</code>
	<code>REAL_4</code>	<code>near_plane</code>
	<code>REAL_4</code>	<code>lod_multiplier</code>
	<code>REAL_4</code>	<code>aspect_ratio</code>
	<code>UNS_2</code>	<code>n_vpts</code>

Returns:            `1 (SUCCEED)`  
                      `0 (FAIL)`

### 2.17.33    `vpt_query.c` (`vpt_vpt_query`)

The `vpt_vpt_query` function copies the viewport parameters node information to a query structure that can be displayed through Gossip. This function is used for testing only.

The function call is `vpt_vpt_query(vp, vi)`, where:

`vp` is a pointer to the viewport parameters  
`vi` is a pointer to the query structure

The function zeroes out the query structure before copying the new data to it.

Called By:            `vptq_vpout`

Routines Called: none

Parameters: VIEWPORT\_PARAMETERS \*vp  
VPT\_VPT\_INFO \*vi

Returns: none

### 2.17.34 vpt\_set.c (vpt\_vpt\_set)

The vpt\_vpt\_set function places new viewport parameter data into the configuration tree. This function is called whenever a node with viewport parameters is defined.

The function call is vpt\_vpt\_set(viewport, node\_index, viewport\_id, database\_id, fov, rez, viewing\_range, near\_plane, lod\_multiplier, aspect\_ratio, paths\_i, paths\_j, start\_path\_id, offset), where:

*viewport* is a pointer to the viewport parameters  
*node\_index* is the unique index used to reference this node  
*viewport\_id* is the identifier assigned to this viewport  
*database\_id* is the database identifier (for future use)  
*fov* is the viewport's field-of-view  
*resolution* is the viewport's screen resolution  
*viewing\_range* is the distance that can be viewed from the viewport  
*near\_plane* is the closest point that can be seen from the viewport  
*lod\_multiplier* is the viewport's level-of-detail multiplier  
*aspect\_ratio* is the ratio of the sides (width:height) of the viewport (for future use)  
*paths\_i* is the number of graphics paths in the i direction  
*paths\_j* is the number of graphics paths in the j direction  
*start\_path\_id* is a pointer to the first graphics path  
*offset* is the offset to the current double buffer (always 0 for initialization)

vpt\_vpt\_set does the following:

- Allocates active area memory for the values required by the DTP (near plane, lod multiplier, and viewing range).
- Fill the viewport node with the parameters passed in.
- Calls vpt\_path\_setup to allocate and fill the graphics paths data.
- Calls vpt\_update to set the final dynamic viewport data.

The function returns SUCCEED if the viewport parameters are added successfully. It returns FAIL if active area memory could not be allocated.

Called By: vpt\_vpt\_process

Routines Called: aam\_malloc  
printf  
vpt\_path\_setup

vpt\_update  
 vpti\_get\_ptr\_cnode  
 vpti\_state\_dvptnode (in debug mode only)

Parameters:	INT_2	node_index
	INT_2	viewport_id
	UNS_1	database_id
	R4IJ	fov
	I4IJ	rez
	REAL_4	viewing_range
	REAL_4	near_plane
	REAL_4	lod_multiplier
	REAL_4	aspect_ratio
	INT_2	paths_i
	INT_2	paths_j
	INT_2	start_path_id
	UNS_4	offset

Returns: 1 (SUCCEED)  
 0 (FAIL)

### 2.17.35 vpt\_update.c

The vpt\_update function finalizes the viewport parameters data, then calls vpt\_path\_process to finalize the viewport's graphics paths data. This function is called for each viewport node defined at initialization time. It is also called whenever a viewport node is updated with the MSG\_VIEW\_MAGNIFICATION message or through Gossip.

The function call is vpt\_update(viewport, fov, rez, viewing\_range, near\_plane, lod\_multiplier, offset), where:

*viewport* is a pointer to the viewport parameters  
*fov* is the viewport's field-of-view  
*rez* is the viewport's screen resolution  
*viewing\_range* is the distance that can be viewed from the viewport  
*near\_plane* is the closest point that can be seen from the viewport  
*lod\_multiplier* is the viewport's level-of-detail multiplier  
*offset* is the offset to the current double buffer

vpt\_update does the following:

- Verifies that the near plane, lod multiplier, and viewing range have been allocated.
- Fills the viewport node.
- Sets the near and far plane values.
- Calculates the level-of-detail multiplier based on the field of view and the horizontal screen resolution.
- Loads the level-of-detail multiplier.
- Calls vpt\_path\_process to set the graphics paths.

The function returns SUCCEED if the viewport node is added successfully. It returns FAIL if no viewport parameters were specified, any of the allocatable values were not allocated, the graphics paths could not be set, or no viewport node was specified.

Called By:	vpt_update_all vpt_update_fov vpt_update_fov_lod vpt_update_lodm vpt_update_near_plane vpt_update_rez vpt_update_view_range vpt_vpt_set	
Routines Called:	printf tan TORAD vpt_path_process	
Parameters:	VIEWPORT_PARAMETERS R4IJ I4IJ REAL_4 REAL_4 REAL_4 REAL_4 UNS_4	*viewport fov rez scale viewing_range near_plane lod_multiplier offset
Returns:	1 (SUCCEED) 0 (FAIL)	



### 2.18 Force Processing (/cig/othersrc/force)

The Force Processing CSC gives the GT100 CIG the ability to display two-dimensional, non-perspective visual data as an overlay on top of the three-dimensional, perspective image of the database. The Force task is the task that runs on the Force board and serves as the data processing interface between the CIG real-time task and the 2-D processor task. The Force board is the physical interface between the VME chassis and the 2-D processor board. This CSC is present on TX backends only.

The real-time software provides 2-D overlay information to the Force board via the Force task. The Force task then writes the data to the GSP, the graphics processor chip on the MPV (Micro Processor Video) board. The GSP contains memory for code storage and for storing and manipulating the 2-D image. The Force board can also read data from GSP memory about particular attributes of the displayed image.

The Force task communicates with the GSP to do the following:

#### **Display the 2-D overlays.**

The original 2-D overlay configuration is passed to Force by the linkup function in the 2-D Overlay Compiler component. The configuration includes the component pointer table, component descriptor table, and window descriptor table. These structures are downloaded into GSP memory and used to generate the overlays displayed on the viewports.

#### **Change the 2-D overlays during runtime.**

Each frame, runtime changes to 2-D components are passed to Force from the real-time software. Each message consists of the command (CHANGE\_DRAW\_2D, DRAW\_TEXT\_2D, ROTATE\_TRANSLATE\_2D, etc.) and any arguments (theta, x translation, y translation, etc.) required for that command. Processing for these messages is as follows:

1. The Simulation Host sends a MSG\_PASS\_ON message to specify the 2-D component changes.
2. The real-time software writes the message to Force memory.
3. The Force task writes the message to GSP memory.
4. The GSP parses each command in the message, updates the component descriptor table in its memory, then regenerates the 2-D overlays.

A new PASS\_ON message is expected every frame. If none is sent, the Force task reprocesses the last PASS\_ON message it received.

The format for each 2-D runtime command is described in the "2-D Commands and Parameters" document.

#### **Return messages to simulation.**

Messages such as error reports can be returned from Force to the Simulation Host. The Force task places the data in Force board memory. The real-time software puts the data into a MSG\_PASS\_BACK message and returns it to the Simulation Host.

**Process laser range request messages.**

The Simulation Host can use the MSG\_REQUEST\_LASER\_RANGE message to request the depth of the pixel located at the screen position represented by i, j, where i is the horizontal coordinate (column) and j is the vertical coordinate (row). The real-time software uses the Force interface to request the pixel depth information from the MPV. The real-time software takes the returned value and sends a MSG\_LASER\_RETURN message to the Simulation Host.

**Process mail.**

This process triggers the Force/MPV interface to send and receive data such as pass on, pass back, and laser range request messages.

**Download and change the 2-D and 3-D color lookup tables.**

The 2-D (final) and 3-D lookup tables are downloaded to the GSP memory through the Force-MPV interface. The active lookup table can be changed using the MSG\_SUBSYS\_MODE message.

**Turn the video channels on and off.**

The MPV video channels can be controlled using the MSG\_VIEWPORT\_UPDATE message.

**Change MPV modes.**

The MPV mode defines the viewport's resolution, panel orientation, and image offset. A viewport's mode can be changed using the MSG\_SUBSYS\_MODE message.

**Start or stop the GSP task.**

The MPV Interface CSC starts the GSP task initially, and stops and restarts it when testing GSP memory. GSP can also be stopped and restarted via Gossip.

**Test reading from/writing to GSP memory.**

GSP memory testing is performed by the MPV Interface CSC at GSP initialization time. Memory testing can also be invoked through Gossip.

The force and GSP tasks are initially loaded and started by functions in the MPV Interface CSC, when the MPV is initialized. The MPV Interface routines and the Force board communicate using two different interface methods: message buffers and the Force front-end control register.

**Message Buffers**

Messages are passed using half-word message buffers. Messages sent from the real-time software (via the MPV Interface routines) to Force are prefixed with MSG\_F0. Messages returned from Force are prefixed with MSG\_F1.

Both the MPV Interface functions and the Force functions use the routines in the mx2\_hword.c CSU to manage and use the message buffers. This CSU is part of both the Real-Time Processing CSC and the Force Processing CSC.

**Force Front-End Control Register**

For some functions, the MPV Interface routines place commands directly into the Force board's front-end control register (FE\_CONTROL). This interface uses an intertask mailbox. The commands are queued in the Force-MPV mailbox until the end of the frame, at which time a message is sent to Force to trigger processing of the queued commands.

Messages passed between the real-time software and the 2-D processor task (on the GSP) are handled in a different manner:

- The Simulation Host sends "pass on" messages to update 2-D overlays during the simulation. The message is received by `process_a_msg`, which calls an MPV Interface routine. The MPV Interface function places the message in an outgoing buffer (`data_buf_out`).
- After processing a "pass on" message, the GSP sends a "pass back" message in reply. The message is placed in an incoming buffer (`data_buf_in`). The MPV Interface routine sets a pointer to the message for return to the Simulation Host.
- The above scenario applies only to changes to the 2-D overlays. The original overlays created before the simulation starts (using the 2-D Overlay Compiler) are not passed through the MPV Interface routines. The 2-D Overlay Compiler's linkup function places the GSP download commands directly into the `FE_CONTROL` register.

The Gossip user can interact with the MPV and the GSP through the `gos_mpvio` function, as follows:

1. The user selects option 4 ("120tx menu") from the Gossip main menu.
2. The user selects option H ("MPV/Force message interface") from the 120tx menu.
3. The Message Interface - RTSW to Force Message menu appears. This menu can be used to send messages to the Force board.

The Force Processing functions used to process messages from the real-time software are named in the format `f0_<message>`. There is a separate function for each message type supported. The functions are defined in the `FG_message_table[]` array, which is defined by the `init_jump_table` function. The `f1_process_messages` function indexes into this array to call the correct function for each message it receives.

The message buffers used as the interface between the real-time software and Force are summarized below. The name shown in quotation marks in the Buffer Name column is the name used in this section to refer to that buffer.

Buffer Name	Purpose/Contents	Messages Pushed By	Messages Popped By
FG_to_force_buf ("to_force buffer")	Messages from the real-time software that affect the simulation.	various libmpvideo functions	f1_process_messages
FG_from_force_buf ("from_force buffer")	Responses to real-time messages.	f0_* functions	various libmpvideo functions
FG_force_query_buf ("force_query buffer")	Messages from Gossip for testing/debugging.	gos_mpvio	f1_process_messages
FG_force_reply_buf ("force_reply buffer")	Responses to Gossip messages.	f0_* functions	gos_mpvio

When an f0\_\* function is called, the calling function passes it a parameter that identifies which outgoing buffer to use for its response messages.

The following table identifies all messages, in alphabetical order, that may be passed from the real-time software or Gossip to Force Processing. The table also identifies the function(s) in the real-time/Gossip software that push the message onto the to\_force or force\_query buffer, and the Force Processing function that processes the message.

RTSW --> Force Message	Sent From (RTSW Function)	Processed By (Force Function)
MSG_F0_3DLUT_DOWNLOAD	gos_mpvio, mpvideo_sim_init	f0_3dlut_download
MSG_F0_3DLUT_SWITCH	gos_mpvio, mpvideo_set_lut	f0_3dlut_switch
MSG_F0_ALLLUT_SWITCH	gos_mpvio, mpvideo_set_lut	f0_alllut_switch
MSG_F0_DEBUG_DISABLE	gos_mpvio	f0_debug_disable
MSG_F0_DEBUG_ENABLE	gos_mpvio	f0_debug_enable
MSG_F0_FINAL_LUT_DOWNLOAD	gos_mpvio, mpvideo_load	f0_final_lut_download
MSG_F0_FINAL_LUT_SWITCH	gos_mpvio, mpvideo_set_lut	f0_final_lut_switch
MSG_F0_MODE_SELECT	gos_mpvio, mpvideo_set_mode	f0_mode_select
MSG_F0_MPV_INIT	gos_mpvio	f0_mpv_init
MSG_F0_MPV_LUT_TYPE_REQUEST	gos_mpvio, mpvideo_boot	f0_mpv_lut_type_request
MSG_F0_MPV_PEEK	gos_mpvio	f0_mpv_peek
MSG_F0_MPV_POKE	gos_mpvio	f0_mpv_poke
MSG_F0_MPV_POKE16	gos_mpvio	f0_mpv_poke16
MSG_F0_MPV_RESET	gos_mpvio, mpvideo_boot	f0_mpv_reset
MSG_F0_MPV_TASK_CONTROL	gos_mpvio	f0_mpv_task_control
MSG_F0_MPV_TEST	gos_mpvio	f0_mpv_test
MSG_F0_MPV_WRITE	gos_mpvio	f0_mpv_write
MSG_F0_PASS_ON	none	f0_pass_on
MSG_F0_PIXEL_DEPTH_REQUEST	gos_mpvio, mpvideo_laser_request_range	f0_pixel_depth_request
MSG_F0_QUERY	gos_mpvio	f0_query
MSG_F0_SET_DISPLAY	gos_mpvio, mpvideo_set_video	f0_set_display
MSG_F0_TRIGGER	gos_mpvio, mpvideo_send_req	f0_trigger

The next table identifies all messages, in alphabetical order, that may be returned from Force to the real-time software or Gossip. The table also identifies the Force function(s) that push the message onto the from\_force or force\_reply buffer, and the function(s) in the real-time/Gossip software that process it.

Force --> RTSW Message	Sent From (Force Function)	Processed By (RTSW Function)
MSG_F1_ACKNOWLEDGE	f0_3dlut_download, f0_3dlut_switch, f0_alllut_switch, f0_debug_disable, f0_debug_enable, f0_final_lut_download, f0_final_lut_switch, f0_mode_select, f0_mpv_poke, f0_mpv_poke16, f0_mpv_reset, f0_mpv_write f0_pass_on, f0_set_display, gsp_io	gos_mpvio, mpvideo_boot, mpvideo_load, mpvideo_response, prtackerr
MSG_F1_MPV_IOCTL	f0_trigger	gos_mpvio, mpvideo_response
MSG_F1_MPV_LUT_TYPE	f0_mpv_lut_type_request	gos_mpvio, mpvideo_boot, mpvideo_response
MSG_F1_MPV_MEMORY	f0_mpv_peek	gos_mpvio, mpvideo_response
MSG_F1_PIXEL_ADDR	f0_pixel_depth_request	gos_mpvio, mpvideo_response
MSG_F1_PIXEL_DEPTH_RETURN	f0_trigger	gos_mpvio, mpvideo_response
MSG_F1_STATUS	none	gos_mpvio, mpvideo_response, prtstaterr
MSG_F1_TEXT	f0_query	gos_mpvio, mpvideo_response

The typedefs that define the structures of all Force-RTSW messages are contained in the mpvideo\_msg.h file.

The following table illustrates the structure of the Force-MPV software mailboxes. This structure is defined in the mpv\_mdef.h file.

Offset	0x7800000 (channel 0)	0x7802000 (channel 1)
0x00000000	MPV_BUFF_IN0	MPV_BUFF_IN1
0x00001000	MPV_BUFF_OUT0	MPV_BUFF_OUT1
0x00001800	PIXEL_REQUEST_0A	PIXEL_REQUEST_1A
0x00001820	PIXEL_REQUEST_0B	PIXEL_REQUEST_1B
0x00001840	PIXEL_I_LOC_0A	PIXEL_I_LOC_1A
0x00001850	PIXEL_J_LOC_0A	PIXEL_J_LOC_1A
0x00001860	PIXEL_DEPTH_0A	PIXEL_DEPTH_1A
0x00001870	PIXEL_I_LOC_0B	PIXEL_I_LOC_1B
0x00001880	PIXEL_J_LOC_0B	PIXEL_J_LOC_1B
0x00001890	PIXEL_DEPTH_0B	PIXEL_DEPTH_1B
0x000018A0	PIXEL_ID_0A	PIXEL_ID_1A
0x000018B0	PIXEL_ID_0B	PIXEL_ID_1B
0x000018C0		
0x000018D0		
0x000018E0	SWITCH_2DLUT_FLAG_0	SWITCH_2DLUT_FLAG_1
0x000018F0	NEW_2DLUT_VALUE_0	NEW_2DLUT_VALUE_1
0x00001900	SWITCH_MODE_FLAG_0	SWITCH_MODE_FLAG_1
0x00001910	NEW_MODE_0	NEW_MODE_1
0x00001920	NEW_HSBLNK_0	NEW_HSBLNK_1
0x00001930	NEW_HEBLNK_0	NEW_HEBLNK_1
0x00001940	NEW_VSBLNK_0	NEW_VSBLNK_1
0x00001950	NEW_VEBLNK_0	NEW_VEBLNK_1

0x07801E00	ILLOP_FLAG
0x07801E10	ILLOP_COUNT
0x07801E20	ILLOP_ADDRESS
0x07801E40	ILLOP_OPCODE

0x07801F00	CLUT_ADDR_PTR
0x07801F20	CLUT_LOADED_FLAG
0x07801F30	
0x07801F40	CLUT_A_SWITCH_FLAG
0x07801F50	CLUT_B_SWITCH_FLAG
0x07801F60	NEW_CLUT_A_ADDR
0x07801F80	NEW_CLUT_B_ADDR
0x07801FA0	CLUT_A_SWITCH_ACK
0x07801FB0	CLUT_B_SWITCH_ACK

Figure 2-22 identifies the CSUs in the Force Processing CSC. The functions performed by these CSUs are described in this section.

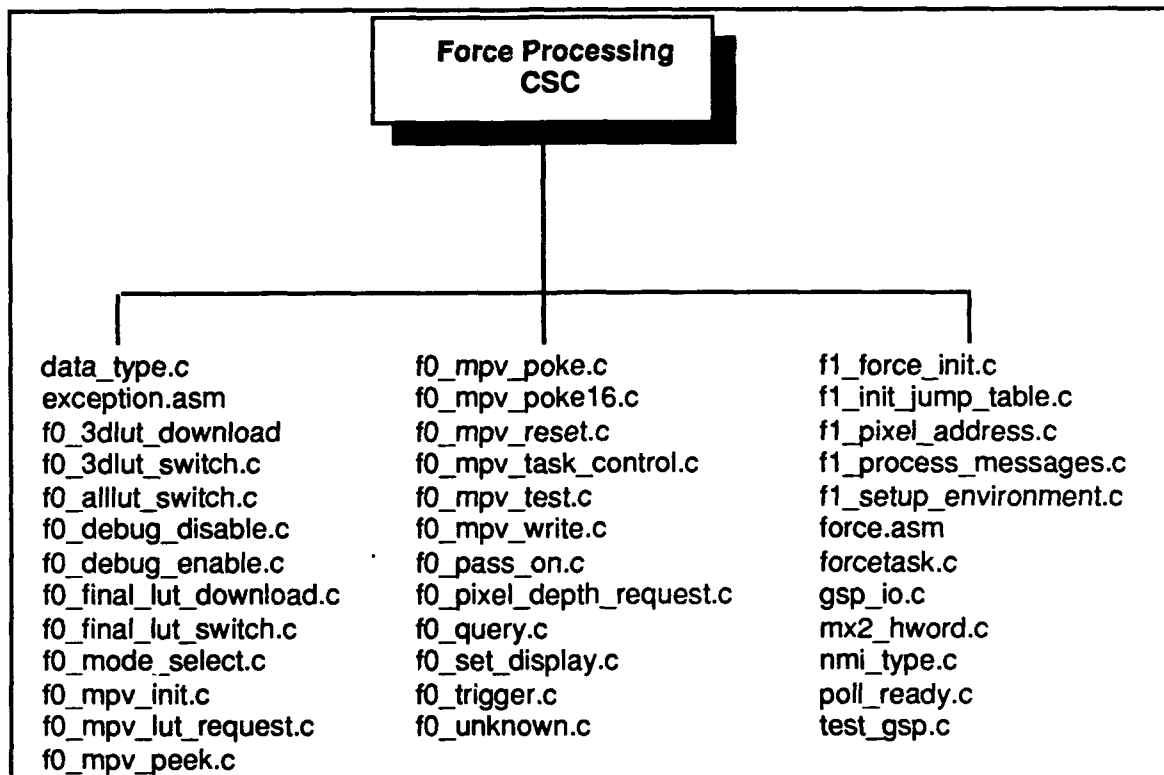


Figure 2-22. Force Processing CSUs

### 2.18.1 data\_type.c

The `data_type` function reads data from and writes data to GSP memory. This function is called by `main` (in `forcetask.c`) if the command in the `FE_CONTROL` register is `SUBSYS_READ_START`, `SUBSYS_WRITE_START`, `SUBSYS_READ_MORE`, or `SUBSYS_WRITE_MORE`.

The function call is `data_type()`. `data_type` does the following:

- Retrieves the type of front-end command: read data or write data.
- If the command is read data:
  - Calls `gsp_ioctl_write` to set the GSP host control word for read mode.
  - Calls `gsp_read` to read the data as specified by the command.
- If the command is write data:
  - Calls `gsp_ioctl_write` to set the GSP host control word for write mode.
  - Calls `gsp_write` to write the data as specified by the command.
  - Reads the data back and verifies it with `compare_buffers`.

Called By:            `main`

Routines Called:   compare\_buffers  
                      gsp\_ioctl\_write  
                      gsp\_read  
                      gsp\_write

Parameters:        none

Returns:           none

### 2.18.2       exception.asm

The functions in the exception.asm CSU are used to initialize the exception vector table. These functions are:

- excep\_init
- spur\_int

#### 2.18.2.1    excep\_init

The excep\_init function first determines the base address of the Force board from the current code. It then initializes the vector base register (VBR) of the 68010 and all entries in the exception vector table to point to spur\_int.

The function call is **excep\_init()**.

Called By:         f1\_force\_init

Routines Called:   spur\_int

Parameters:        none

Returns:           none

#### 2.18.2.2    spur\_int

The spur\_int function disables interrupts, then saves all of the 68010 data registers into the structure "context." The order of the save is as follows: D0-D7, A0-A6, SSP, USP, PC, SR.

The function call is **spur\_int()**.

Called By:         excep\_init



Routines Called: none

Parameters: none

Returns: none

### 2.18.3 f0\_3dlut\_download.c

The `f0_3dlut_download` function processes the `MSG_F0_3DLUT_DOWNLOAD` message. This message is sent by `mpvideo_sim_init` at the beginning of a simulation to download 16 3-D color lookup tables. The message can also be sent through Gossip by selecting the 1 ("3D\_LUT\_DOWNLOAD") option from the `gos_mpvio` menu. After the tables are downloaded, one can be put into effect using the `MSG_F0_3DLUT_SWITCH` or `MSG_F0_ALLLUT_SWITCH` message.

The function call is `f0_3dlut_download(message_P, buffer_P)`, where:

*message\_P* is a pointer to the `MSG_F0_3DLUT_DOWNLOAD` message

*buffer\_P* is a pointer to the buffer to be used for the response message

`f0_3dlut_download` does the following:

- Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer with the acknowledgement code set to 0.
- Calls `gsp_ioctl_write` to set the GSP host control word for read mode.
- Calls `gsp_read` to read the color lookup tables into Force memory.
- If the 16 tables are found:
  - Uses `gsp_write` to download the 3-D color lookup tables from Force memory into the 2-D task tables in GSP memory.
  - Verifies the data back with `gsp_read`.
  - Calls `gsp_write` to write the lookup table loaded flag to GSP memory.
  - Sets the `FG_3dlut_downloaded_flag` to TRUE.
  - Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer, with the acknowledgement code set to 1 (success).
- If the tables are not found:
  - Sets `FG_3dlut_downloaded_flag` to FALSE.
  - Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer, with the acknowledgement code set to 0xFFFF (error).
- Calls `gsp_write` to write the lookup table switch acknowledgement flags to GSP memory.
- Calls `gsp_ioctl_write` to set the GSP host control word for write mode.
- Calls `gsp_write` to download the tables to the appropriate addresses in the MPV.

The function returns the *status* received from `mx2_push`.

Called By: `f1_process_messages` (through `*FG_message_table`)

Routines Called: `gsp_ioctl_write`

gsp\_read  
gsp\_write  
mx2\_push

Parameters:           UNS\_4                           \*message\_P  
                          UNS\_4                           \*buffer\_P

Returns:               status

#### 2.18.4       f0\_3dlut\_switch.c

The f0\_3dlut\_switch function processes the MSG\_F0\_3DLUT\_SWITCH message. This message is used to put a specified 3-D color lookup table into effect. (The table must have already been downloaded to the GSP.) The message is sent by mpvideo\_set\_lut if the Simulation Host sends a MSG\_SUBSYS\_MODE message that specifies only a new 3-D table. The message can also be sent through Gossip by selecting the 2 ("3D\_LUT\_SWITCH") option from the gos\_mpvio menu.

The function call is **f0\_3dlut\_switch(message\_P, buffer\_P)**, where:

*message\_P* is a pointer to the MSG\_F0\_3DLUT\_SWITCH message  
*buffer\_P* is a pointer to the buffer to be used for the response message

f0\_3dlut\_switch does the following:

- If the FG\_3dlut\_downloaded\_flag is TRUE (3-D color lookup tables have been downloaded by f0\_3dlut\_download):
  - Calls gsp\_ioctl\_write to set the GSP host control word for write mode.
  - Calls gsp\_write to write the new table address in the GSP and set the GSP's switch flag for the specified channel.
  - Calls gsp\_write to write the appropriate acknowledgement flag to the GSP.
  - Pushes a MSG\_F1\_ACKNOWLEDGE message onto the response buffer, with the acknowledgement code set to the new table id (success)..
- If the FG\_3dlut\_downloaded\_flag is FALSE (f0\_3dlut\_download did not download tables):
  - Pushes a MSG\_F1\_ACKNOWLEDGE message onto the response buffer with the acknowledgement code set to 0xFFFF (error).

The function always returns 0.

Called By:           f1\_process\_messages (through \*FG\_message\_table)

Routines Called:    gsp\_ioctl\_write  
                          gsp\_write  
                          mx2\_push

Parameters:           MSG\_F0\_3DLUT\_SWITCH           \*message\_P  
                          UNS\_4                           \*buffer\_P

Returns: 0

### 2.18.5 f0\_alllut\_switch.c

The `f0_alllut_switch` function processes the `MSG_F0_ALLLUT_SWITCH` message. This message is used to put both a new 3-D lookup table and a new final (2-D) lookup table into effect. The 3-D table must have already been downloaded to the GSP. The message is sent by `mpvideo_set_lut` if the Simulation Host sends a `MSG_SUBSYS_MODE` message that specifies both a new 3-D table and a new 2-D table. The message can also be sent through Gossip by selecting the 3 ("ALL\_LUT\_SWITCH") option from the `gos_mpvio` menu.

The function call is `f0_alllut_switch(message_P, buffer_P)`, where:

*message\_P* is a pointer to the `MSG_F0_ALLLUT_SWITCH` message  
*buffer\_P* is a pointer to the buffer to be used for the response message

`f0_alllut_switch` does the following:

- Calls `gsp_ioctl_read` to read the GSP host control word.
- Calls `gsp_ioctl_write` to modify the host control word to set the data strobe bit on; this signals the GSP that I/O is required.
- Determines the MPV buffer base address from the host control word.
- Calls `gsp_ioctl_write` to set the host control word for write mode.
- If `FG_3dlut_downloaded_flag` is TRUE (`f0_3dlut_download` has downloaded 3-D color tables):
  - Gets the new 3-D table id from the message.
  - Finds the table in Force memory.
  - Finds the correct address in GSP memory.
  - Calls `gsp_write` to download the table to GSP memory.
- If `FG_3dlut_downloaded_flag` is FALSE (`f0_3dlut_download` has not downloaded 3-D color tables):
  - Sets the 3-D table id to 0x00FF (error).
- Gets the new 2-D table id from the message.
- Calls `gsp_write` to download the new 2-D table to GSP memory.
- Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer; the acknowledgement code is set to the new 2-D table value OR'd with the new 3-D table value (which is 0x00FF if 3-D tables have not been downloaded to the GSP).

The function always returns 0.

Called By: `f1_process_messages` (through `*FG_message_table`)

Routines Called: `gsp_ioctl_read`  
`gsp_ioctl_write`  
`gsp_write`  
`mx2_push`

Parameters:	MSG_F0_ALLLUT_SWITCH UNS_4	*message_P *buffer_P
-------------	-------------------------------	-------------------------

Returns:	0
----------	---

### 2.18.6 f0\_debug\_disable.c

The `f0_debug_disable` function processes the `MSG_F0_DEBUG_DISABLE` message. This message is used to disable Force debug mode. The message can be generated through Gossip by selecting the L ("DEBUG DISABLE") option on the `gos_mpvio` menu.

At the current time, the Force debug flag is used only by `gsp_io`. If debug is enabled, `gsp_io` returns a status message.

The function call is `f0_debug_disable(message_P, buffer_P)`, where:

*message\_P* is a pointer to the `MSG_F0_DEBUG_DISABLE` message  
*buffer\_P* is a pointer to the buffer to be used for the response message

`f0_debug_disable` does the following:

- Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer, with the acknowledgement code set to 0 (success).
- Sets the `FG_debug_flag` variable to FALSE.

The function returns the *status* received from `mx2_push`.

Called By:	f1_process_messages (through *FG_message_table)
------------	---

Routines Called:	mx2_push
------------------	----------

Parameters:	UNS_4 UNS_4	*message_P *buffer_P
-------------	----------------	-------------------------

Returns:	status
----------	--------

### 2.18.7 f0\_debug\_enable.c

The `f0_debug_enable` function processes the `MSG_F0_DEBUG_ENABLE` message. This message is used to enable Force debug mode. The message is generated through Gossip by selecting the K ("DEBUG ENABLE") option from the `gos_mpvio` menu.

At the current time, the Force debug flag is used only by `gsp_io`. If debug is enabled, `gsp_io` returns a status message.

The function call is `f0_debug_enable(message_P, buffer_P)`, where:

*message\_P* is a pointer to the MSG\_F0\_DEBUG\_ENABLE message  
*buffer\_P* is a pointer to the buffer to be used for the response message

f0\_debug\_enable does the following:

- Pushes a MSG\_F1\_ACKNOWLEDGE message onto the response buffer, with the acknowledgement code set to 0 (success).
- Sets the FG\_debug\_flag variable to TRUE.

The function returns the *status* received from mx2\_push.

Called By: f1\_process\_messages (through \*FG\_message\_table)

Routines Called: mx2\_push

Parameters: UNS\_4 \*message\_P  
 UNS\_4 \*buffer\_P

Returns: status

#### 2.18.8 f0\_final\_lut\_download.c

The f0\_final\_lut\_download function processes the MSG\_F0\_FINAL\_LUT\_DOWNLOAD message. This message is used to download the final (2-D) color lookup table to the GSP. The message is sent by mpvideo\_load if the Simulation Host sends a MSG\_FILE\_DESCR message that specifies a final lookup table. The message can also be generated through Gossip by selecting the 4 ("FINAL\_LUT\_DWNLD") option from the gos\_mpvio menu.

The function call is f0\_final\_lut\_download(message\_P, buffer\_P), where:

*message\_P* is a pointer to the MSG\_F0\_FINAL\_LUT\_DOWNLOAD message  
*buffer\_P* is a pointer to the buffer to be used for the response message

f0\_final\_lut\_download does the following:

- If FG\_final\_lut\_type is 0 (2-D table cannot be downloaded to this MPV board — final lookup table is stored in PROM):
  - Pushes a MSG\_F1\_ACKNOWLEDGE message onto the response buffer, with the acknowledgement code set to 0xFFFF (error).
- If the entry number in the message is greater than or equal to 192 (the maximum allowed is 191):
  - Pushes a MSG\_F1\_ACKNOWLEDGE message onto the response buffer, with the acknowledgement code set to 0xFFFE (error).
- If the FG\_final\_lut\_type is 1 (this MPV board has downloadable memory for lookup tables) and the entry number is valid:
  - Finds the specified entry's table and address offsets.
  - Finds the appropriate destination address in the MPV.
  - For each two bytes of data in the message (maximum 256, or 512 bytes):
    - \* Calls gsp\_write to write the data to the GSP.

- \* Calls `gsp_read` to read the data back to compare it.
- Calls `gsp_ioctl_write` to set the GSP host control word for write mode.
- Calls `gsp_write` to download all of the data from the message to GSP memory.
- Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer, with the acknowledgement code set to the entry number from the message.

The function always returns 0.

Called By:	<code>f1_process_messages</code> (through <code>*FG_message_table</code> )	
Routines Called:	<code>gsp_read</code> <code>gsp_write</code> <code>mx2_push</code>	
Parameters:	<code>MSG_F0_FINAL_LUT_DOWNLOAD</code> <code>UNS_4</code>	<code>*message_P</code> <code>*buffer_P</code>
Returns:	0	

#### 2.18.9 `f0_final_lut_switch.c`

The `f0_final_lut_switch` function processes the `MSG_F0_FINAL_LUT_SWITCH` message. This message is used to put a specified final (2-D) color lookup table into effect. The message is sent by `mpvideo_set_lut` if the Simulation Host sends a `MSG_SUBSYS_MODE` message that specifies only a new 2-D table. The message can also be generated through Gossip by selecting the 5 ("FINAL\_LUT\_SWITCH") option from the `gospvpio` menu.

The function call is `f0_final_lut_switch(message_P, buffer_P)`, where:

*message\_P* is a pointer to the `MSG_F0_FINAL_LUT_SWITCH` message  
*buffer\_P* is a pointer to the buffer to be used for the response message

`f0_final_lut_switch` does the following:

- Calls `gsp_ioctl_read` to read the GSP host control word.
- Calls `gsp_ioctl_write` to change the GSP host control word to set the data strobe bit on; this signals the GSP that I/O is required.
- Determines the MPV buffer base address from the host control word.
- Calls `gsp_ioctl_write` to set the GSP host control word for write mode.
- Gets the new 2-D table id (value) from the message.
- Calls `gsp_write` to download the new 2-D table to GSP memory.
- Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer, with the acknowledgement code set to the new 2-D lut value.

The function always returns 0.

Called By: `f1_process_messages` (through `*FG_message_table`)

Routines Called: `gsp_ioctl_read`  
`gsp_ioctl_write`  
`gsp_write`  
`mx2_push`

Parameters: `MSG_F0_FINAL_LUT_SWITCH` `*message_P`  
`UNS_4` `*buffer_P`

Returns: 0

### 2.18.10 `f0_mode_select.c`

The `f0_mode_select` function processes the `MSG_F0_MODE_SELECT` message. This message is used to put the desired resolution and panel orientation into effect. The message is sent by `mpvideo_set_mode` to load the default mode when the MPV is initialized, and later to change the mode if the Simulation Host sends a `MSG_SUBSYS_MODE` message. The message can also be generated through Gossip by selecting the 6 ("MODE\_SELECT") option from the `gos_mpvio` menu.

This message is supported only with the Value-Added MPV board (revision 5).

The function call is `f0_mode_select(message_P, buffer_P)`, where:

*message\_P* is a pointer to the `MSG_F0_MODE_SELECT` message  
*buffer\_P* is a pointer to the buffer to be used for the response message

`f0_mode_select` does the following:

- If `FG_mpv_type` is 0 (older MPV board):
  - Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer with the acknowledgement code set to 0xFFFF (error).
- If `FG_mpv_type` is 1 (Value-Added MPV board):
  - Retrieves the new mode, orientation, resolution *i* and *j*, and offset *i* and *j* from the message.
  - Puts the data into the `mode_select_data` structure.
  - Calls `gsp_ioctl_read` to read the GSP host control word.
  - Calls `gsp_ioctl_write` to modify the host control word to set the data strobe bit on; this signals the GSP that I/O is required.
  - Determines the MPV buffer base address from the host control word.
  - Calls `gsp_ioctl_write` to set the host control word for write mode.
  - Calls `gsp_write` to write the new values from the `mode_select_data` structure into MPV memory.
  - Calls `f1_pa_new_orientation` to change the viewport's orientation.
  - Calls `f1_pa_new_resolution` to change the viewport's resolution.
  - Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer, with the acknowledgement code set to the 2-D lookup table value written into the MPV board's configuration register.

The function returns the *status* received from *mx2\_push* if successful. It returns 0 if the *mpv\_type* is 0.

Called By: *f1\_process\_messages* (through *\*FG\_message\_table*)

Routines Called: *mx2\_push*  
*gsp\_ioctl\_read*  
*gsp\_ioctl\_write*  
*gsp\_write*  
*f1\_pa\_new\_orientation*  
*f1\_pa\_new\_resolution*

Parameters: *MSG\_F0\_MODE\_SELECT* *\*message\_P*  
*UNS\_4* *\*buffer\_P*

Returns: 0  
*status*

#### 2.18.11 *f0\_mpv\_init.c*

The *f0\_mpv\_init* function processes the *MSG\_F0\_MPV\_INIT* message. This message is used to initialize the MPV. The message can be generated through Gossip by selecting the 7 ("MPV\_INIT") option from the *gos\_mpvio* menu.

The function call is *f0\_mpv\_init(message\_P, buffer\_P)*, where:

*message\_P* is a pointer to the *MSG\_F0\_MPV\_INIT* message

*buffer\_P* is a pointer to the buffer to be used for the response message

This function is not currently implemented. It always returns 0.

Called By: *f1\_process\_messages* (through *\*FG\_message\_table*)

Routines Called: none

Parameters: *UNS\_4* *\*message\_P*  
*UNS\_4* *\*buffer\_P*

Returns: 0

#### 2.18.12 *f0\_mpv\_lut\_type\_request.c*

The *f0\_mpv\_lut\_type\_request* function processes the *MSG\_F0\_MPV\_LUT\_TYPE\_REQUEST* message. This message is used to determine the MPV's board type and



whether or not the board is capable of accepting downloaded final (2-D) lookup tables. The function returns the data in a MSG\_F1\_MPV\_LUT\_TYPE message.

The MSG\_F0\_MPV\_LUT\_TYPE\_REQUEST message is sent by mpvideo\_boot when the system is initialized. The message can also be generated through Gossip by selecting the 8 ("MPV\_LUT\_TYPE\_REQ") option from the gos\_mpvio menu.

The function call is **f0\_mpv\_lut\_type\_request(message\_P, buffer\_P)**, where:

*message\_P* is a pointer to the MSG\_F0\_MPV\_LUT\_TYPE\_REQUEST message  
*buffer\_P* is a pointer to the buffer to be used for the response message

f0\_mpv\_lut\_type\_request does the following:

- Gets the MPV type from the FG\_mpv\_type global variable (1=Value-Added MPV board, 0=older MPV board).
- Gets the final lookup table type from the FG\_final\_lut\_type global variable (1=final lookup table can be downloaded, 0=final lookup table is in PROM).
- Pushes a MSG\_F1\_MPV\_LUT\_TYPE message containing the above parameters onto the response buffer.

The function returns the *status* received from mx2\_push.

Called By: f1\_process\_messages (through \*FG\_message\_table)

Routines Called: mx2\_push

Parameters: UNS\_4 \*message\_P  
 UNS\_4 \*buffer\_P

Returns: status

### 2.18.13 f0\_mpv\_peek.c

The f0\_mpv\_peek function processes the MSG\_F0\_MPV\_PEEK message. This message is used to read data from GSP memory. The message specifies the starting address and the number of bytes to read. The data is returned in a MSG\_F1\_MPV\_MEMORY message.

The MSG\_F0\_MPV\_PEEK message can be generated through Gossip by selecting the 9 ("MPV\_PEEK") option from the gos\_mpvio menu.

The function call is **f0\_mpv\_peek(message\_P, buffer\_P)**, where:

*message\_P* is a pointer to the MSG\_F0\_MPV\_PEEK message  
*buffer\_P* is a pointer to the buffer to be used for the response message

f0\_mpv\_peek does the following:

- Converts the byte count specified in the message to a half-word count (maximum 256 half-words, for 512 bytes).
- Calls `gsp_ioctl_write` to set the GSP host control word for read mode.
- Calls `gsp_read` to read the specified amount of data from GSP memory, starting at the specified address.
- Pushes a `MSG_F1_MPV_MEMORY` message with the data read from the GSP onto the response buffer.

The function always returns 0.

Called By: `f1_process_messages` (through `*FG_message_table`)

Routines Called: `gsp_ioctl_write`  
`gsp_read`  
`mx2_push`

Parameters: `MSG_F0_MPV_PEEK` `*message_P`  
`UNS_4` `*buffer_P`

Returns: 0

#### 2.18.14 `f0_mpv_poke.c`

The `f0_mpv_poke` function processes the `MSG_F0_MPV_POKE` message. This message is used to write one 32-bit value to GSP memory. The message specifies the starting address and the data to be written. The message can be generated through Gossip by selecting the A ("MPV\_POKE") option from the `gos_mpvio` menu.

The function call is `f0_mpv_poke(message_P, buffer_P)`, where:

*message\_P* is a pointer to the `MSG_F0_MPV_POKE` message

*buffer\_P* is a pointer to the buffer to be used for the response message

`f0_mpv_poke` does the following:

- Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer, with the acknowledgement code set to 0 (success).
- Calls `gsp_ioctl_write` to set the GSP host control word for write mode.
- Calls `gsp_write` to write the data from the message to the specified location.

The function returns the *status* received from `mx2_push`.

Called By: `f1_process_messages` (through `*FG_message_table`)

Routines Called: `gsp_ioctl_write`  
`gsp_write`  
`mx2_push`

Parameters:           MSG\_F0\_MPV\_POKE           \*message\_P  
                          UNS\_4                   \*buffer\_P

Returns:               status

### 2.18.15     f0\_mpv\_poke16.c

The `f0_mpv_poke16` function processes the `MSG_F0_MPV_POKE16` message. This message is used to write one 16-bit value to GSP memory. The message specifies the starting address and the data to be written. The message can be generated through Gossip by selecting the **M** ("MPV POKE16") option from the `gos_mpvio` menu.

The function call is `f0_mpv_poke16(message_P, buffer_P)`, where:

*message\_P* is a pointer to the `MSG_F0_MPV_POKE16` message  
*buffer\_P* is a pointer to the buffer to be used for the response message

`f0_mpv_poke16` does the following:

- Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer, with the acknowledgement code set to 0 (success).
- Calls `gsp_write` to write the data from the message to the specified location.

The function returns the *status* received from `mx2_push`.

Called By:            f1\_process\_messages (through \*FG\_message\_table)

Routines Called:     gsp\_write  
                         mx2\_push

Parameters:           MSG\_F0\_MPV\_POKE           \*message\_P  
                          UNS\_4                   \*buffer\_P

Returns:              status

### 2.18.16     f0\_mpv\_reset.c

The `f0_mpv_reset` function processes the `MSG_F0_MPV_RESET` message. This message is used to reset the MPV board. The message is sent by `mpvideo_boot` when the system is initialized if the MPV board is detected to be a Value-Added MPV. The message can also be generated through Gossip by selecting the **B** ("MPV\_RESET") option from the `gos_mpvio` menu.

The function call is `f0_mpv_reset(message_P, buffer_P)`, where:

*message\_P* is a pointer to the MSG\_F0\_MPV\_RESET message  
*buffer\_P* is a pointer to the buffer to be used for the response message

f0\_mpv\_reset does the following:

- Calls gsp\_reset to reset the MPV board.
- Sets nmi\_set\_flag to 0 (this indicates that the GSP is not running).
- Pushes a MSG\_F1\_ACKNOWLEDGE reply message onto the response buffer, with the acknowledgement code set to 0 (success).

The function returns the *status* received from mx2\_push.

Called By: f1\_process\_messages (through \*FG\_message\_table)

Routines Called: gsp\_reset  
mx2\_push

Parameters: UNS\_4 \*message\_P  
UNS\_4 \*buffer\_P

Returns: status

#### 2.18.17 f0\_mpv\_task\_control.c

The f0\_mpv\_task\_control function processes the MSG\_F0\_MPV\_TASK\_CONTROL message. This message is used to start or stop the MPV task. The message can be generated through Gossip by selecting the C ("MPV\_TASK\_CTL") option from the gos\_mpvio menu.

The function call is f0\_mpv\_task\_control(message\_P, buffer\_P), where:

*message\_P* is a pointer to the MSG\_F0\_MPV\_TASK\_CONTROL message  
*buffer\_P* is a pointer to the buffer to be used for the response message

This function is not currently implemented. It always returns 0.

Called By: f1\_process\_messages (through \*FG\_message\_table)

Routines Called: none

Parameters: UNS\_4 \*message\_P  
UNS\_4 \*buffer\_P

Returns: 0

### 2.18.18 f0\_mpv\_test.c

The `f0_mpv_test` function processes the `MSG_F0_MPV_TEST` message. This message is used to request that the Force task perform MPV/Force interface tests and report the results. The message can be generated through Gossip by selecting the **D** ("MPV\_TEST") option from the `gos_mpvio` menu.

The function call is `f0_mpv_test(message_P, buffer_P)`, where:

*message\_P* is a pointer to the `MSG_F0_MPV_TEST` message

*buffer\_P* is a pointer to the buffer to be used for the response message

This function is not currently implemented. It always returns 0.

Called By: `f1_process_messages` (through `*FG_message_table`)

Routines Called: none

Parameters: `UNS_4` `*message_P`  
`UNS_4` `*buffer_P`

Returns: 0

### 2.18.19 f0\_mpv\_write.c

The `f0_mpv_write` function processes the `MSG_F0_MPV_WRITE` message. This message is used to write up to 512 bytes of data to GSP memory. The message specifies the data to be written and the starting destination address. The message can be generated through Gossip by selecting the **E** ("MPV\_WRITE") option from the `gos_mpvio` menu.

The function call is `f0_mpv_write(message_P, buffer_P)`, where:

*message\_P* is a pointer to the `MSG_F0_MPV_WRITE` message

*buffer\_P* is a pointer to the buffer to be used for the response message

`f0_mpv_write` does the following:

- Converts the byte count in the message to a half-word count.
- Calls `gsp_ioctl_write` to set the GSP host control word for write mode.
- Calls `gsp_write` to write the data from the message to the specified address in GSP memory.
- Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer, with the acknowledgement code set to 0 (success).

The function always returns 0.

Called By: f1\_process\_messages (through \*FG\_message\_table)

Routines Called: gsp\_ioctl\_write  
gsp\_write  
mx2\_push

Parameters: MSG\_F0\_MPV\_WRITE \*message\_P  
UNS\_4 \*buffer\_P

Returns: 0

### 2.18.20 f0\_pass\_on.c

The f0\_pass\_on function processes the MSG\_F0\_PASS\_ON message. This message is sent when the Simulation Host sends messages that are to be forwarded to an imbedded subsystem. Currently, the only subsystem that receives pass\_on messages is the 2-D processor task. The message is used to update 2-D overlay components (stored in GSP memory) during runtime.

The MSG\_F0\_PASS\_ON message is not currently used. For real-time PASS\_ON messages, the msg\_pass\_on function (in the Message Processing CSC) puts the message data in a buffer and the Force task writes it to GSP memory. The MSG\_F0\_PASS\_ON message cannot be generated through Gossip either, as the F ("PASS\_ON") option on the gos\_mpvio menu is not currently implemented.

The function call is f0\_pass\_on(message\_P, buffer\_P), where:

*message\_P* is a pointer to the MSG\_F0\_PASS\_ON message  
*buffer\_P* is a pointer to the buffer to be used for the response message

f0\_pass\_on does the following:

- Reads and saves the GSP host control word.
- Calls gsp\_ioctl\_write to modify the host control word to set the data strobe bit on; this signals the GSP that I/O is required.
- Gets the MPV buffer base address from the control word.
- Calls gsp\_ioctl\_write to set the host control word for write mode.
- Calls gsp\_write to write the new data to GSP memory.
- Pushes a MSG\_F1\_ACKNOWLEDGE message onto the response buffer, with the acknowledgement code set to 0 (success).
- Calls gsp\_ioctl\_write to restore the original control word, turning the data strobe bit off.

The function always returns 0.

Called By: f1\_process\_messages (through \*FG\_message\_table)

Routines Called: gsp\_ioctl\_read

gsp\_ioctl\_write  
gsp\_write  
mx2\_push

Parameters: MSG\_F0\_PASS\_ON                      \*message\_P  
                 UNS\_4                                \*buffer\_P

Returns: 0

### 2.18.21 f0\_pixel\_depth\_request.c

The `f0_pixel_depth_request` function processes the `MSG_F0_PIXEL_DEPTH_REQUEST` message. This message is used to obtain a depth value for a specified pixel location. The message is sent by `mpvideo_laser_request_range` when the Simulation Host sends a `MSG_LASER_REQUEST_RANGE` message for a TX backend. The message can also be generated through Gossip by selecting the G ("PIXEL\_DEPTH\_REQ") option from the `gos_mpvio` menu.

`f0_pixel_depth_request` takes the data from the `MSG_F0_PIXEL_DEPTH_REQUEST` message and puts it into several arrays that are used by `f0_trigger` to download the request to the MPV.

The function call is `f0_pixel_depth_request(message_P, buffer_P)`, where:

*message\_P* is a pointer to the `MSG_F0_PIXEL_DEPTH_REQUEST` message  
*buffer\_P* is a pointer to the buffer to be used for the response message

`f0_pixel_depth_request` does the following:

- Gets the channel number, pixel location (i,j), and range id from the message.
- If the channel number is greater than 1, returns 0 and terminates.
- Calls `f1_pa_fb_offset` to get the pixel location's frame buffer address offset.
- Sets the channel's element in the `FG_pixel_addr[]` array to the pixel's GSP address. `f0_trigger` uses this value when sending the request to the MPV.
- Sets the channel's element in the `FG_pixel_id[]` array to the range id from the message. `f0_trigger` uses this value when sending the request to the MPV.
- Sets the channel's element in the `FG_pixel_depth_flag[]` array to TRUE. This prompts `f0_trigger` to read the depth data returned from the MPV and generate a `MSG_F1_PIXEL_DEPTH_RETURN` message.
- Pushes a `MSG_F1_PIXEL_ADDR` message onto the response buffer.
- Sets the channel's element in the `FG_new_pixel_req_flag[]` array to 4. This tells `f0_trigger` that there is a depth request to send to the MPV.
- Sets the `FG_pixel_depth_message` flag to TRUE. This tells `gsp_io` that pixel locations have been set.

The function always returns 0.

Called By: f1\_process\_messages (through \*FG\_message\_table)

Routines Called:     f1\_pa\_fb\_offset  
                      mx2\_push

Parameters:           MSG\_F0\_PIXEL\_DEPTH\_REQUEST     \*message\_P  
                      UNS\_4                            \*buffer\_P

Returns:              0

### 2.18.22     f0\_query.c

The f0\_query.c CSU contains the function used to process the MSG\_F0\_QUERY message. It also contains various functions used to manipulate character strings. The functions in this CSU are:

- f0\_query
- strcpy
- strcat
- strlen

#### 2.18.22.1   f0\_query

The f0\_query function processes the MSG\_F0\_QUERY message. This message is used to transfer Gossip data to and request general information from the Force task. The message can be generated through Gossip by selecting the I ("QUERY") option from the gos\_mpvio menu. The Gossip user specifies the query code and text.

The function call is **f0\_query(message\_P)**, where *message\_P* is a pointer to the MSG\_F0\_QUERY message.

f0\_query does the following:

- Selects a predefined string based on the query code in the message.
- Copies the string.
- Concatenates the string and the text provided in the message.
- Puts the concatenated string in the reply message.
- Pushes a MSG\_F1\_TEXT message onto the force\_reply buffer.

The function always returns 1.

Called By:            f1\_process\_messages (through \*FG\_message\_table)

Routines Called:     mx2\_push  
                      strcat  
                      strcpy

Parameters:           MSG\_F0\_QUERY                    \*message\_P



Returns: 1

### 2.18.22.2 strcpy

The strcpy function copies one string to another.

The function call is **strcpy(string1, string2)**, where:

*string1* is the destination string  
*string2* is the source string

Called By: f0\_query

Routines Called: none

Parameters: char \*string1  
char \*string2

Returns: none

### 2.18.22.3 strcat

The strcat routine concatenates a string onto the end of another string.

The function call is **strcat(string1, string2)**, where:

*string1* is the first string  
*string2* is the string to be added to the end of *string1*

The concatenated string is stored in *string1*.

Called By: f0\_query

Routines Called: strlen

Parameters: char \*string1  
char \*string2

Returns: none

#### 2.18.22.4 strlen

**The strlen function determines the length of a given string.**

The function call is **strlen(string)**, where *string* is a pointer to the string. The string length is returned as *index*.

**Called By:** `strcat`

Routines Called: none

**Parameters:**           char                                 \*string

Returns: index

### 2.18.23 f0 set display.c

The `f0_set_display` function processes the `MSG_F0_SET_DISPLAY` message. This message is used to turn the video channels driven by the MPV board on and off. The message is sent by `mpvideo_set_video` at the beginning of a simulation (to turn all channels on) and at the end of a simulation (to turn all channels off). It is also sent during a simulation if the Simulation Host sends a `MSG_VIEWPORT_UPDATE` message. The message can also be generated through Gossip by selecting the `H` ("SET\_DISPLAY") option from the `gos_mpvio` menu.

The function call is **f0 set display(message P, buffer P)**, where:

*message* *P* is a pointer to the MSG\_F0\_SET\_DISPLAY message

*buffer P* is a pointer to the buffer to be used for the response message

**f0\_set\_display** does the following:

- Calls `gsp_ioctl_write` to set the GSP host control word for write mode.
- If the message specifies `MPV_VIDEO_ON`:
  - Sets the `mpv_video_code` to ON.
  - Calls `gsp_write` to write the `mpv_video_code` to the video control address.
- If the message specifies `MPV_VIDEO_OFF`:
  - Sets the `mpv_video_code` to OFF.
  - Calls `gsp_write` to write the `mpv_video_code` to the video control address.
- Pushes a `MSG_F1_ACKNOWLEDGE` message onto the response buffer, with the acknowledgement code set to the `mpv_video_code`.

**The function always returns 0.**

**Called By:** f1\_process\_messages (through \*FG\_message\_table)

Routines Called:    gsp\_ioctl\_write  
                      gsp\_write  
                      mx2\_push

Parameters:        MSG\_F0\_SET\_DISPLAY        \*message\_P  
                      UNS\_4                    \*buffer\_P

Returns:            0

#### 2.18.24    f0\_trigger.c

The `fo_trigger` function processes the `MSG_F0_TRIGGER` message. This message triggers the MPV I/O to communicate directly with the MPV board to process all requests queued during the preceding frame. The function returns a `MSG_F1_PIXEL_DEPTH` message for each channel for which laser processing has been requested. It also returns a `MSG_F1_MPV_IOCTL` message if an illegal opcode trap is detected on the MPV.

The `MSG_F0_TRIGGER` message is sent by `mpvideo_send_req` every frame, after all of the frame's messages have been processed by the real-time software. The message can also be generated through Gossip by selecting the J ("TRIGGER") option from the `gos_mpvio` menu.

The MPV locations for each channel's pixel depth request and return data are specified in terms of their offset from the MPV board's base address. The offsets are defined in the `mpv_mdef.h` file.

The function call is `f0_trigger(message_P, buffer_P)`, where:

*message\_P* is a pointer to the `MSG_F0_TRIGGER` message  
*buffer\_P* is a pointer to the buffer to be used for the response message

`f0_trigger` does the following:

- Calls `gsp_ioctl_read` to read the GSP host control word.
- Calls `gsp_ioctl_write` to modify the host control word to set the data strobe bit on; this signals the GSP that I/O is required.
- Determines the MPV buffer base address from the host control word.
- Calls `gsp_ioctl_write` to set the host control word for read mode.
- If `FG_pixel_depth_flag[0]` is TRUE, indicating that channel 0 has a laser depth request pending:
  - Sets the channel number to 0.
  - Calls `gsp_read` to read the pixel depth for channel 0 (located at the MPV's base address plus the offset specified by `PIXEL_DEPTH_A_OFF`).
  - Calls `gsp_read` to read the range id for this request (located at the MPV's base address plus the offset specified by `PIXEL_ID_A_OFF`).
  - Pushes a `MSG_F1_PIXEL_DEPTH_RETURN` message for channel 0 onto the response buffer.
- If `FG_pixel_depth_flag[1]` is TRUE, indicating that channel 1 has a laser depth request pending:
  - Sets the channel number to 1.

- Calls `gsp_read` to read the pixel depth for channel 1 (located at the MPV's base address plus the offset specified by `PIXEL_DEPTH_B_OFF`).
- Calls `gsp_read` to read the range id for this request (located at the MPV's base address plus the offset specified by `PIXEL_ID_B_OFF`).
- Pushes a `MSG_F1_PIXEL_DEPTH_RETURN` message for channel 1 onto the response buffer.
- Calls `gsp_ioctl_write` to set the host control word for write mode.
- If `FG_new_pixel_req_flag[0]` is greater than 0 (indicating that channel 0 has a new pixel depth request message to send to the MPV; `f0_pixel_depth_request` sets this value to 4):
  - Decrements the value in `FG_new_pixel_req_flag[0]`.
  - Calls `gsp_write` to write the pixel address to the MPV (located at the MPV's based address plus the offset in `PIXEL_REQUEST_A_OFF`).
  - Calls `gsp_write` to write the pixel id to the MPV (at the MPV's base address plus the offset in `PIXEL_ID_A_OFF`).
- If `FG_new_pixel_req_flag[1]` is greater than 0 (indicating that channel 1 has a new pixel depth request message to send to the MPV; `f0_pixel_depth_request` sets this value to 4):
  - Decrements the value in `FG_new_pixel_req_flag[1]`.
  - Calls `gsp_write` to write the pixel address to the MPV (located at the MPV's based address plus the offset in `PIXEL_REQUEST_B_OFF`).
  - Calls `gsp_write` to write the pixel id to the MPV (at the MPV's base address plus the offset in `PIXEL_ID_B_OFF`).
- Calls `gsp_ioctl_write` to set the host control word for read mode.
- Calls `gsp_read` to check for any illegal opcodes from the GSP.
- If any opcode errors were found, pushes a `MSG_F1_MPV_IOCTL` onto the response buffer.

The function returns the *status* received from `mx2_push`.

Called By: `f1_process_messages` (through `*FG_message_table`)

Routines Called: `gsp_ioctl_read`  
`gsp_ioctl_write`  
`gsp_read`  
`gsp_write`  
`mx2_push`

Parameters: `UNS_4` `*message_P`  
`UNS_4` `*buffer_P`

Returns: `status`

### 2.18.25 `f0_unknown.c`

The `f0_unknown` function is called if an unknown message type is received from the real-time software. The function does no processing.

The function call is `f0_unknown(message_P, buffer_P)`, where:

*message\_P* is a pointer to the unknown message  
*buffer\_P* is a pointer to the buffer to be used for the response message

The function always returns 0.

Called By: f1\_process\_messages (through \*FG\_message\_table)

Routines Called: none

Parameters: UNS\_4 \*message\_P  
 UNS\_4 \*buffer\_P

Returns: 0

#### 2.18.26 f1\_force\_init.c

The f1\_force\_init function initializes the Force task upon initial (cold) start.

The function call is f1\_force\_init(). The function does the following:

- Calls excep\_init to initialize the exception vector table.
- Initializes the Force interface mailbox.
- Initializes the Force control register in the Force interface mailbox for GSP I/O (as opposed to test mode in which the GSP is not actually read or written to).
- Initializes the sync registers in the Force interface mailbox.
- Calls init\_ports to initialize the Force-GSP interface.
- Calls f1\_pa\_init to initialize the pixel address object.
- Calls gsp\_read to determine the MPV resolution (from the GSP configuration register).
- Calls f1\_pa\_new\_resolution to set the MPV resolution (RES\_640x480 or RES\_320x240).
- Initializes the FG\_final\_lut\_type global variable to MPV\_NO\_LUT\_DWNLD (2-D lookup table is not downloadable because table is contained in PROM).
- Initializes the FG\_pixel\_depth\_message global variable to FALSE (no locations have been set for laser range processing).
- Sets the FG\_mpv\_type global variable based on the setting of the MPV DIP switch: 1 for Value-Added MPV boards, 0 for older boards.
- If the MPV type is 1 (Value-Added MPV), sets FG\_final\_lut\_type to MPV\_LUT\_DWNLD (2-D lookup table can be downloaded to the MPV).
- Calls f1\_setup\_environment to set up the Force environment variables to the real-time software.
- Calls f1\_init\_jump\_table to initialize the array used to call the appropriate f0\_\* function for each message received from the real-time software.

Called By: main

Routines Called:    `excep_init`  
                     `f1_init_jump_table`  
                     `f1_pa_init`  
                     `f1_pa_new_resolution`  
                     `f1_setup_environment`  
                     `gsp_read`  
                     `init_ports`

Parameters:        `none`

Returns:           `none`

### 2.18.27    `f1_init_jump_table.c`

The `f1_init_jump_table` function initializes the Force message processing jump table (`FG_message_table`). This array contains one entry for each valid message type that can be received from the real-time software. Each entry in the array contains the name of the `f0_*` function to call for that message type. The `f1_process_messages` function indexes into `FG_message_table[]` to call the appropriate `f0_*` function for each message it receives.

The function call is `f1_init_jump_table()`. The function first initializes all 255 entries to point to the `f0_unknown` function, then modifies the entries for all valid message types.

The index number assigned to each valid message type is defined in the `mpvideo_msg.h` file.

Called By:         `f1_force_init`

Routines Called:   `none`

Parameters:        `none`

Returns:           `none`

### 2.18.28    `f1_pixel_address.c`

The functions in the `f1_pixel_address.c` CSU are used to return the frame buffer address offset for a given pixel location. The CSU contains a separate function for each resolution and orientation. These functions are:

- `f1_pa_init`
- `f1_pa_new_orientation`
- `f1_pa_new_resolution`
- `f1_pa_fb_offset`
- `f1_pa_640x480_v`
- `f1_pa_320x240_v`

- f1\_pa\_640x256\_v
- f1\_pa\_640x240\_v
- f1\_pa\_640x480\_h
- f1\_pa\_320x240\_h
- f1\_pa\_640x256\_h
- f1\_pa\_640x240\_h

The pa\_FP\_table array contains one element for each supported orientation/resolution. Each element contains the name of the f1\_pa\_\*\_v or f1\_pa\_\*\_h function used to process frame buffer address requests for viewports with the specified attributes.

The f1\_pa\_\*\_v and f1\_pa\_\*\_h functions are called via the \*pixel\_address\_FP function pointer. This pointer is set by indexing into pa\_FP\_table[] with a specified orientation and resolution.

#### 2.18.28.1 f1\_pa\_init

The f1\_pa\_init function initializes the pa\_FP\_table array when the Force board is initialized. pa\_FP\_table[] contains one entry for each supported orientation/resolution combination. Each entry contains the name of the f1\_pa\_\*\_v or f1\_pa\_\*\_h function that processes frame buffer address requests for viewports with that orientation and resolution.

The function call is f1\_pa\_init(). The function does the following:

- Defines each element in the pa\_FP\_table array.
- Sets the default viewport orientation to vertical.
- Sets the default viewport resolution to 640 x 480.
- Sets the \*pixel\_address\_FP function pointer to the default orientation and resolution. This function pointer is used by other Force functions to index into the pa\_FP\_table for a given orientation and resolution.

The function always returns 1.

Called By: f1\_force\_init

Routines Called: none

Parameters: none

Returns: 1

#### 2.18.28.2 f1\_pa\_new\_orientation

The f1\_pa\_new\_orientation function changes the orientation of a viewport. This function is called to load the default orientation when the MPV is initialized, and later to change the orientation if the Simulation Host sends a MSG\_SUBSYS\_MODE message. A new orientation can also be specified through Gossip. Orientation changes are supported only by the Value-Added MPV board.

The function call is **f1\_pa\_new\_orientation(new\_orientation)**, where *new\_orientation* is ORIENT\_VERTICAL or ORIENT\_HORIZONTAL.

f1\_pa\_new\_orientation does the following:

- Verifies that the new orientation is valid.
- Resets the \*pixel\_address\_FP function pointer to reflect the new orientation.

The function returns 1 if successful. It returns 0 if the new orientation is invalid.

Called By:	f0_mode_select	
Routines Called:	none	
Parameters:	HWND	new_orientation
Returns:	0 1	

#### 2.18.28.3 f1\_pa\_new\_resolution

The f1\_pa\_new\_resolution function changes the resolution of a viewport. This function is called to load the default resolution when the MPV is initialized, and later to change the resolution if the Simulation Host sends a MSG\_SUBSYS\_MODE message. A new resolution can also be specified through Gossip. Resolution changes are supported only by the Value-Added MPV board.

The function call is **f1\_pa\_new\_resolution(new\_resolution)**, where *new\_resolution* is RES\_640x480, RES\_320x240, RES\_640x256, or RES\_640x240.

f1\_pa\_new\_resolution does the following:

- Verifies that the new resolution is valid.
- Resets the \*pixel\_address\_FP function pointer to reflect the new resolution.

The function returns 1 if successful. It returns 0 if the new resolution is invalid.

Called By:	f0_mode_select f1_force_init	
Routines Called:	none	
Parameters:	HWND	new_resolution



Returns:           0  
                  1

#### 2.18.28.4 f1\_pa\_fb\_offset

The f1\_pa\_fb\_offset function finds the frame buffer address offset for a given pixel location on a specified viewport. This function is called to process laser range requests.

The function call is **f1\_pa\_fb\_offset(pixel\_i, pixel\_j, viewport)**, where:

*pixel\_i* is the horizontal coordinate of the pixel  
*pixel\_j* is the vertical coordinate of the pixel  
*viewport* is the viewport number (0 or 1)

f1\_pa\_fb\_offset uses the \*pixel\_address\_FP function pointer to call the appropriate f1\_pa\*\_v or f1\_pa\*\_h function to determine the pixel's frame buffer address offset. It then returns this value as *fb\_offset*.

Called By:           f0\_pixel\_depth\_request  
                      gsp\_io

Routines Called:    \*pixel\_address\_FP

Parameters:	HWND	pixel_i
	HWND	pixel_j
	HWND	viewport

Returns:           fb\_offset

#### 2.18.28.5 f1\_pa\_640x480\_v

The f1\_pa\_640x480\_v function returns the frame buffer address offset for a given pixel location for viewports with 640 x 480 resolution and vertical panel orientation.

The function call is **f1\_pa\_640x480\_v(pixel\_i, pixel\_j, viewport)**, where:

*pixel\_i* is the horizontal coordinate of the pixel  
*pixel\_j* is the vertical coordinate of the pixel  
*viewport* is the viewport number (always 0)

f1\_pa\_640x480\_v does the following:

- Verifies that the viewport number is 0 and the pixel location is valid (pixel\_i < 640 and pixel\_j < 480).
- Sets the panel offset (panel starting address) based on the specified horizontal offset, and reduces the horizontal offset accordingly.
- Computes the total offset by OR'ing the panel, horizontal, and vertical offsets.

The function returns the frame buffer address offset (*gsp\_total\_offset*) if successful. It returns 0 if the viewport number or pixel location is invalid.

Called By:            f1\_pa\_fb\_offset            (through \*pixel\_address\_FP)

Routines Called:    none

Parameters:	HWND	pixel_i
	HWND	pixel_j
	HWND	viewport

Returns:            0  
                      gsp\_total\_offset

#### 2.18.28.6    f1\_pa\_320x240\_v

The f1\_pa\_320x240\_v function returns the frame buffer address offset for a given pixel location for viewports with 320 x 240 resolution and vertical panel orientation.

The function call is **f1\_pa\_320x240\_v(pixel\_i, pixel\_j, viewport)**, where:

*pixel\_i* is the horizontal coordinate of the pixel  
*pixel\_j* is the vertical coordinate of the pixel  
*viewport* is the viewport number (0 or 1)

f1\_pa\_320x240\_v does the following:

- Verifies that the viewport number is either 0 or 1 and the pixel location is valid (pixel\_i < 320 and pixel\_j < 240).
- Sets the panel offset (panel starting address) based on the specified horizontal offset, and reduces the horizontal offset accordingly.
- Modifies the panel offset if working with viewport 1.
- Computes the total offset by OR'ing the panel, horizontal, and vertical offsets.

The function returns the frame buffer address offset (*gsp\_total\_offset*) if successful. It returns 0 if the viewport number or pixel location is invalid.

Called By:            f1\_pa\_fb\_offset            (through \*pixel\_address\_FP)

Routines Called:    none

Parameters:	HWND	pixel_i
	HWND	pixel_j
	HWND	viewport

Returns:                   0  
                          gsp\_total\_offset

#### 2.18.28.7 f1\_pa\_640x256\_v

The f1\_pa\_640x256\_v function returns the frame buffer address for a given pixel location for viewports with 640 x 256 resolution and vertical panel orientation.

The function call is **f1\_pa\_640x256\_v(pixel\_i, pixel\_j, viewport)**, where:

*pixel\_i* is the horizontal coordinate of the pixel  
*pixel\_j* is the vertical coordinate of the pixel  
*viewport* is the viewport number (0 or 1)

f1\_pa\_640x256\_v does the following:

- Verifies that the viewport number is either 0 or 1 and the pixel location is valid (*pixel\_i* < 640 and *pixel\_j* < 256).
- Sets the panel offset (panel starting address) based on the specified horizontal offset, and reduces the horizontal offset accordingly.
- Modifies the panel offset if working with viewport 1.
- Computes the total offset by OR'ing the panel, horizontal, and vertical offsets.

The function returns the frame buffer address offset (*gsp\_total\_offset*) if successful. It returns 0 if the viewport number or pixel location is invalid.

Called By:                   f1\_pa\_fb\_offset                   (through \*pixel\_address\_FP)

Routines Called:           none

Parameters:	HWND	pixel_i
	HWND	pixel_j
	HWND	viewport

Returns:                   0  
                          gsp\_total\_offset

#### 2.18.28.8 f1\_pa\_640x240\_v

The f1\_pa\_640x240\_v function returns the frame buffer address for a given pixel location for viewports with 640 x 240 resolution and vertical orientation.

The function call is **f1\_pa\_640x240\_v(pixel\_i, pixel\_j, viewport)**, where:

*pixel\_i* is the horizontal coordinate of the pixel  
*pixel\_j* is the vertical coordinate of the pixel  
*viewport* is the viewport number (always 0)

**f1\_pa\_640x240\_v** does the following:

- Verifies that the viewport number is 0 and the pixel location is valid (*pixel\_i* < 640 and *pixel\_j* < 240).
- Sets the panel offset (panel starting address) based on the specified horizontal offset, and reduces the horizontal offset accordingly.
- Computes the total offset by OR'ing the panel, horizontal, and vertical offsets.

The function returns the frame buffer address offset (*gsp\_total\_offset*) if successful. It returns 0 if the viewport number or pixel location is invalid.

Called By:            **f1\_pa\_fb\_offset**            (through *\*pixel\_address\_FP*)

Routines Called:    **none**

Parameters:	<b>HWND</b>	<b>pixel_i</b>
	<b>HWND</b>	<b>pixel_j</b>
	<b>HWND</b>	<b>viewport</b>

Returns:            **0**  
                      **gsp\_total\_offset**

#### **2.18.28.9    f1\_pa\_640x480\_h**

The **f1\_pa\_640x480\_h** function returns the frame buffer address for a given pixel location for viewports with 640 x 480 resolution and horizontal panel orientation.

The function call is **f1\_pa\_640x480\_h(pixel\_i, pixel\_j, viewport)**, where:

*pixel\_i* is the horizontal coordinate of the pixel  
*pixel\_j* is the vertical coordinate of the pixel  
*viewport* is the viewport number (always 0)

**f1\_pa\_640x480\_h** does the following:

- Verifies that the viewport number is 0 and the pixel location is valid (*pixel\_i* < 640 and *pixel\_j* < 480).
- Sets the panel offset (panel starting address) based on the specified vertical offset, and reduces the vertical offset accordingly.
- Computes the total offset by OR'ing the panel, horizontal, and vertical offsets.

The function returns the frame buffer address offset (*gsp\_total\_offset*) if successful. It returns 0 if the viewport number or pixel location is invalid.

Called By:            **f1\_pa\_fb\_offset**            (through *\*pixel\_address\_FP*)

Routines Called:    **none**

Parameters:	HWORDB HWORDB HWORDB	pixel_i pixel_j viewport
-------------	----------------------------	--------------------------------

Returns:	0 gsp_total_offset
----------	-----------------------

**2.18.28.10 f1\_pa\_320x240\_h**

The `f1_pa_320x240_h` function returns the frame buffer address for a given pixel location for viewports with 320 x 240 resolution and horizontal panel orientation.

The function call is `f1_pa_320x240_h(pixel_i, pixel_j, viewport)`, where:

*pixel\_i* is the horizontal coordinate of the pixel  
*pixel\_j* is the vertical coordinate of the pixel  
*viewport* is the viewport number (0 or 1)

`f1_pa_320x240_h` does the following:

- Verifies that the viewport number is either 0 or 1 and the pixel location is valid (`pixel_i < 320` and `pixel_j < 240`).
- Sets the panel offset (panel starting address) based on the specified vertical offset, and reduces the vertical offset accordingly.
- Modifies the panel offset if working with viewport 1.
- Computes the total offset by OR'ing the panel, horizontal, and vertical offsets.

The function returns the frame buffer address offset (*gsp\_total\_offset*) if successful. It returns 0 if the viewport number or pixel location is invalid.

Called By:	f1_pa_fb_offset	(through *pixel_address_FP)
------------	-----------------	-----------------------------

Routines Called:	none
------------------	------

Parameters:	HWORDB HWORDB HWORDB	pixel_i pixel_j viewport
-------------	----------------------------	--------------------------------

Returns:	0 gsp_total_offset
----------	-----------------------

**2.18.28.11 f1\_pa\_640x256\_h**

The `f1_pa_640x256_h` function returns the frame buffer address for a given pixel location for viewports with 640 x 256 resolution and horizontal panel orientation.

The function call is **f1\_pa\_640x256\_h(pixel\_i, pixel\_j, viewport)**, where:

*pixel\_i* is the horizontal coordinate of the pixel  
*pixel\_j* is the vertical coordinate of the pixel  
*viewport* is the viewport number (0 or 1)

**f1\_pa\_640x256\_h** does the following:

- Verifies that the viewport number is either 0 or 1 and the pixel location is valid (*pixel\_i* < 640 and *pixel\_j* < 256).
- Sets the panel offset (panel starting address) based on the specified vertical offset, and reduces the vertical offset accordingly.
- Modifies the panel offset if working with viewport 1.
- Computes the total offset by OR'ing the panel, horizontal, and vertical offsets.

The function returns the frame buffer address offset (*gsp\_total\_offset*) if successful. It returns 0 if the viewport number or pixel location is invalid.

Called By:                    **f1\_pa\_fb\_offset**                    (through \**pixel\_address\_FP*)

Routines Called:            **none**

Parameters:	<b>HWND</b>	<b>pixel_i</b>
	<b>HWND</b>	<b>pixel_j</b>
	<b>HWND</b>	<b>viewport</b>

Returns:                    **0**  
                               **gsp\_total\_offset**

#### **2.18.28.12 f1\_pa\_640x240\_h**

The **f1\_pa\_640x240\_h** function returns the frame buffer address for a given pixel location for viewports with 640 x 240 resolution and horizontal orientation.

The function call is **f1\_pa\_640x240\_h(pixel\_i, pixel\_j, viewport)**, where:

*pixel\_i* is the horizontal coordinate of the pixel  
*pixel\_j* is the vertical coordinate of the pixel  
*viewport* is the viewport number (always 0)

**f1\_pa\_640x240\_h** does the following:

- Verifies that the viewport number is 0 and the pixel location is valid (*pixel\_i* < 640 and *pixel\_j* < 240).
- Sets the panel offset (panel starting address) based on the specified vertical offset, and reduces the vertical offset accordingly.
- Computes the total offset by OR'ing the panel, horizontal, and vertical offsets.

The function returns the frame buffer address offset (*gsp\_total\_offset*) if successful. It returns 0 if the viewport number or pixel location is invalid.

Called By:            *f1\_pa\_fb\_offset*            (through *\*pixel\_address\_FP*)

Routines Called:    none

Parameters:	HWORDB	<i>pixel_i</i>
	HWORDB	<i>pixel_j</i>
	HWORDB	<i>viewport</i>

Returns:            0  
                      *gsp\_total\_offset*

### 2.18.29    *f1\_process\_messages.c*

The *f1\_process\_messages* function handles the messages that are sent to Force from the real-time software and Gossip. It previews each message in both incoming message buffers (*to\_force* and *force\_query*), and calls the appropriate *f0\_\** function to process it.

The function call is *f1\_process\_messages()*. The function does the following:

- Calls *mx2\_peek* to preview the top message in the *to\_force* buffer (used for messages from the real-time software).
- If a message is found (indicated by a status of *MX\_MESSAGE\_PREVIEWED* from *mx2\_push*):
  - Calls *se\_clock* to read the Force timers and store the values.
  - Turns on the Force lights.
  - Indexes into the *\*FG\_message\_table* with the message code; this invokes the appropriate *f0\_\** routine to process the message. Specifies *FG\_from\_force\_buf* as the response buffer to be used by the *f0\_\** routine.
  - Calls *mx2\_skip* to remove the processed message from the *to\_force* buffer.
  - Turns off the Force lights.
  - Calls *se\_clock* to read the Force timers and store the values.
- Calls *mx2\_peek* to preview the top message in the *force\_query* buffer (used for messages from Gossip).
- If a message is found (indicated by a status of *MX\_MESSAGE\_PREVIEWED* from *mx2\_push*):
  - Calls *se\_clock* to read the Force timers and store the values.
  - Turns on the Force lights.
  - Indexes into the *\*FG\_message\_table* with the message code; this invokes the appropriate *f0\_\** routine to process the message. Specifies *FG\_force\_reply\_buf* as the response buffer to be used by the *f0\_\** routine.
  - Calls *mx2\_skip* to remove the processed message from the *force\_query* buffer.
  - Turns off the Force lights.
  - Calls *se\_clock* to read the Force timers and store the values.

Called By: poll\_ready

Routines Called: \*FG\_message\_table  
mx2\_peek  
mx2\_skip  
se\_clock

Parameters: none

Returns: none

### 2.18.30 f1\_setup\_environment.c

The f1\_setup\_environment function sets up the environment variables for the Force software. This function is called when the Force board is initialized.

The function call is f1\_setup\_environment(). The function does the following:

- Sets up the global variables for the message buffers.
- Sets up the Force environment variables for the Force-CIG type, the software version, and the size of each message buffer.
- Sets the force\_started variable to TRUE; this tells the real-time software that the environment setup is complete.
- Calls mx2\_open to open the Force-RTSW and Force-Gossip message buffers.

The function always returns TRUE.

Called By: f1\_force\_init

Routines Called: mx2\_open

Parameters: none

Returns: TRUE

### 2.18.31 force.asm

The force.asm CSU contains a group of subroutines used by the Force functions to read from and write to the GSP. These functions are the following:

- init\_ports
- gsp\_write
- gsp\_read
- gsp\_ioctl\_read
- gsp\_ioctl\_write



- `gsp_reset`

This module is written in assembly language to obtain optimal performance from the 68230-to-GSP interface.

### 2.18.31.1 `init_ports`

The `init_ports` function is called at start-up to initialize the Force-GSP interface. The function call is `init_ports()`.

Called By: `f1_force_init`

Routines Called: `none`

Parameters: `none`

Returns: `none`

### 2.18.31.2 `gsp_write`

The `gsp_write` function writes a block of data from the Force board's memory down to the GSP. This function is used to download subsystem mode data, color lookup tables, 2-D overlay data, and pixel addresses and ids for laser range processing.

The function call is `gsp_write(number_hwords, data_buffer, gsp_address)`, where:

*number\_hwords* is the number of words to be written to the GSP  
*data\_buffer* is the location of the data in Force memory  
*gsp\_address* is the address to write to

Called By: `data_type`  
`f0_3dlut_download`  
`f0_3dlut_switch`  
`f0_alllut_switch`  
`f0_final_lut_download`  
`f0_final_lut_switch`  
`f0_mode_select`  
`f0_mpv_poke`  
`f0_mpv_poke16`  
`f0_mpv_write`  
`f0_pass_on`  
`f0_set_display`  
`f0_trigger`  
`gsp_io`  
`main`  
`nmi_type`

	poll_ready test_gsp	
Routines Called:	none	
Parameters:	HWND HWND WORD	number_hwords *data_buffer gsp_address
Returns:	none	

### 2.18.31.3 gsp\_read

The `gsp_read` function reads a block of data from the GSP into Force memory. This function is used to read color lookup tables, pixel depths and ids, and the MPV's resolution. It is also used to check for illegal opcode errors and to verify writes to the GSP.

The function call is `gsp_read (number_hwords, data_buffer, gsp_address)`, where:

*number\_hwords* is the number of words to be read from the GSP  
*data\_buffer* is the location of the data in Force memory  
*gsp\_address* is the address to read from

Called By:	data_type f0_3dlut_download f0_final_lut_download f0_mpv_peek f0_trigger f1_force_init gsp_io main test_gsp	
Routines Called:	none	
Parameters:	HWND HWND WORD	number_hwords *data_buffer gsp_address
Returns:	none	

#### 2.18.31.4 gsp\_ioctl\_read

The `gsp_ioctl_read` function reads the control word from the GSP host interface control register. This function is used to read the control word before modifying it to set the data strobe bit on to signal the GSP of required I/O.

The function call is `gsp_ioctl_read()`. The function returns the control word as an integer (half word = 16 bits).

Called By:            `f0_alllut_switch`  
                      `f0_final_lut_switch`  
                      `f0_mode_select`  
                      `f0_pass_on`  
                      `f0_trigger`  
                      `gsp_io`  
                      `main`

Routines Called:    `none`

Parameters:         `none`

Returns:            `control_data`

#### 2.18.31.5 gsp\_ioctl\_write

The `gsp_ioctl_write` function writes the control word to the GSP host interface control register. This function is used to set the control word for read or write mode, or to set the data strobe bit on to signal the GSP of required I/O.

The function call is `gsp_ioctl_write(control_data)`, where *control\_data* is the control word to be written.

Called By:            `data_type`  
                      `f0_3dlut_download`  
                      `f0_3dlut_switch`  
                      `f0_alllut_switch`  
                      `f0_final_lut_switch`  
                      `f0_mode_select`  
                      `f0_mpv_peek`  
                      `f0_mpv_poke`  
                      `f0_mpv_write`  
                      `f0_pass_on`  
                      `f0_set_display`  
                      `f0_trigger`  
                      `gsp_io`  
                      `main`

nmi\_type  
poll\_ready  
test\_gsp

Routines Called: none

Parameters: int control\_data

Returns: none

#### 2.18.31.6 gsp\_reset

The gsp\_reset function resets the Value-Added MPV board. This function is called when the system is initialized if a Value-Added MPV is detected. The function can also be invoked through Gossip by selecting the B ("MPV\_RESET") option on the gos\_mpvio menu.

The function call is gsp\_reset().

Called By: f0\_mpv\_reset

Routines Called: none

Parameters: none

Returns: none

#### 2.18.32 forcetask.c

The forcetask.c CSU contains the main Force Processing task and various functions related to timing. The functions in this CSU are:

- main
- compare\_buffers
- restart\_clock
- red\_clock
- se\_clock

##### 2.18.32.1 main

The main function processes commands placed in the Force front-end control register by the real-time software and Gossip. This task is loaded and started by the bootforce function in the MPV Interface CSC.

The function call is **main()**. **main** does the following:

- Calls **fl\_force\_init** to initialize the Force control registers, various parameters, and the Force-GSP interface.
- Turns off the Force lights.
- Calls **poll\_ready** to read the command in the **FE\_CONTROL** register.
- Processes each message, calling other Force functions as appropriate.
- Clears the ready bit in the **FE\_CONTROL** register.

The following table describes the processing performed by **main** for each command sent by **linkup** or **gos\_mpvio**. The first column identifies the command, preceded by the upper byte of the value in the **FE\_CONTROL** register (this is the value returned by **poll\_ready**). The second column describes the purpose of the command (in *italics*), then shows the major steps performed by **main** to process that command.

Message	Processing by main
0 SUBSYS_MAIL_SEND	<i>Process mail, pixel depth information, and pass_on 2-D components to/from the GSP.</i> Makes sure <b>nmi_set_flag</b> is 1 (triggers GSP I/O processing); turns Force lights on; calls <b>se_clock</b> ; calls <b>gsp_io</b> ; calls <b>se_clock</b> ; turns Force lights off.
1 SUBSYS_READ_START, SUBSYS_WRITE_START, SUBSYS_READ_MORE, SUBSYS_WRITE_MORE	<i>Send data to or receive data from the GSP.</i> Turns Force lights on; calls <b>se_clock</b> ; calls <b>data_type</b> ; calls <b>se_clock</b> ; turns Force lights off.
2 SUBSYS_NMI_START	<i>Start the GSP task.</i> Calls <b>nmi_type</b> .
3 SUBSYS_TEST_MEM	<i>Test the ability to read from/write to GSP memory.</i> Calls <b>test_gsp</b> ; if any errors are returned, sets Force control register.
6 SUBSYS_STOP	<i>Halt the GSP task.</i> Calls <b>gsp_ioctl_write</b> to set halt mode; sets <b>nmi_set_flag</b> to 0.
10 SUBSYS_READ_HCTRL	<i>Read the control register.</i> Calls <b>gsp_ioctl_read</b> ; puts response in <b>data_buf_in</b> .
11 SUBSYS_WRITE_HCTRL	<i>Write to the control register.</i> Calls <b>gsp_ioctl_write</b> to write data from <b>data_buf_out</b> .
12 SUBSYS_READ_HDATA	<i>Read data from GSP memory.</i> Calls <b>gsp_read</b> to read data into <b>data_buf_in</b> .
13 SUBSYS_WRITE_HDATA	<i>Write data to GSP memory.</i> Calls <b>gsp_ioctl_write</b> to set write mode; calls <b>gsp_write</b> to write data from <b>data_buf_out</b> ; calls <b>gsp_ioctl_write</b> to set read mode; calls <b>gsp_read</b> and compare_buffers to verify data was written correctly.
15	<i>Reset Force clock.</i> Calls <b>restart_clock</b> .

Called By: none

Routines Called:     compare\_buffers  
                      data\_type  
                      fl\_force\_init  
                      gsp\_io  
                      gsp\_ioctl\_read  
                      gsp\_ioctl\_write  
                      gsp\_read  
                      gsp\_write  
                      nmi\_type  
                      poll\_ready  
                      red\_clock  
                      restart\_clock  
                      se\_clock  
                      test\_gsp

Parameters:           none

Returns:              none

#### 2.18.32.2   compare\_buffers

The compare\_buffers function is a boolean function that compares the contents of two buffers. This function is used to verify writes to GSP memory.

The function call is **compare\_buffer(hword\_count, ptr1, ptr2)**, where:

*hword\_count* is the length of the data to be compared  
*ptr1* and *ptr2* are pointers to the buffers to be compared

The function returns TRUE if the buffer contents are equal, and FALSE if they are not.

Called By:            data\_type  
                      main

Routines Called:     none

Parameters:           HWORD                           hword\_count  
                      HWORD                           \*ptr1  
                      HWORD                           \*ptr2

Returns:              1 (TRUE)  
                      0 (FALSE)

### 2.18.32.3 restart\_clock

The restart\_clock function resets the Force clocks to 0.

The function call is **restart\_clock()**.

Called By: main

Routines Called: none

Parameters: none

Returns: none

### 2.18.32.4 red\_clock

The red\_clock function reads the Force clocks and places the values read into the locations passed in the call.

The function call is **red\_clock(timerval, pittimer)**, where:

*timerval* is a pointer to real-time clock

*pittimer* is a pointer to pit clock

Called By: se\_clock

Routines Called: none

Parameters: UNS\_2 \*timerval  
UNS\_4 \*pittimer

Returns: none

### 2.18.32.5 se\_clock

The se\_clock function triggers the Force clock to be read and the values placed into a timing array. The timing array, *timet[]*, is indexed by a code supplied by the calling function. Each element in the array contains the real-time and pit timer values returned by red\_clock.

The function call is **se\_clock(code)**, where *code* is the index (0-511) into the *timet[]* array. The function does the following:

- Sets the *code* to 0 if it is greater then 511.
- Calls `red_clock` to read the Force timers and place the values in the element specified by *code* in the `timit[]` array.

Called By: f1\_process\_messages  
main  
poll\_ready

**Routines Called:** red\_clock

Parameters: UNS\_2 code

**Returns:** none

### 2.18.33 gsp\_io.c

The `gsp_io` function processes mail and pixel depth data to and from the GSP. This function is called when the command in the Force front-end control register is `SUBSYS_MAIL_SEND`.

The function call is **gsp\_io()**. **gsp\_io** does the following:

- Calls `gsp_ioctl_read` to read the control word.
- Calls `gsp_ioctl_write` to modify the host control word to set the data strobe bit on; this signals the GSP that I/O is required.
- Determines the buffer id and base address from the host control word.
- If the `FG_debug_flag` is on, pushes a `MSG_F1_ACKNOWLEDGE` message onto the `force_reply` buffer to return status.
- Gets the data buffer size from `SUB_DATA_BUFF`.
- Calls `gsp_ioctl_write` to set the host control word for write mode.
- Calls `gsp_write` to write the SIM-to-2D data to 2D memory.
- If `FG_pixel_depth_message` is `FALSE` (no pixel locations selected for laser range processing):
  - Calls `f1_pa_fb_offset` to get the GSP address offsets of the default pixel location for each channel.
  - Calls `gsp_write` to write the pixel address offsets to GSP memory.
- Calls `gsp_ioctl_write` to set the host control word for read mode.
- Calls `gsp_read` to put the data from the 2D-to-SIM buffer into Force memory.
- Calls `gsp_ioctl_write` to set the host control word for read mode.
- Calls `gsp_read` to put the pixel depth return data for both channels into Force memory.
- Calls `gsp_ioctl_write` to restore the original control word, turning the data strobe bit off.
- Clears the ready bit in the front-end control register to signal the real-time software that it is done.

Called By: main



Routines Called:     fl\_pa\_fb\_offset  
                      gsp\_ioctl\_read  
                      gsp\_ioctl\_write  
                      gsp\_read  
                      gsp\_write  
                      mx2\_push

Parameters:           none

Returns:               none

### 2.18.34     mx2\_hword.c

The functions in the mx2\_hword.c CSU manage the half-word message buffers used for the interface between Force Processing and the real-time software. These functions are:

- mx2\_open
- mx2\_push
- mx2\_peek
- mx2\_skip
- mx2\_error
- mx2\_hwcopy

These functions are also present in the Real-Time Processing CSC (see section 2.13). The Routines Called and Called By lists in this section do not include any real-time software (e.g., MPV Interface) functions. Similarly, the Routines Called and Called By lists in section 2.13 do not include any Force Processing functions.

The mx2\_hword.c file also contains the typedef for the message buffer (MX2\_DEVICE) and message header (MESSAGE\_HEADER) structures.

#### 2.18.34.1   mx2\_open

The mx2\_open function initializes a half-word message buffer with a specified size.

The function call is **mx2\_open(dev\_P, device\_size)**, where:

*dev\_P* is a pointer to the message buffer  
*device\_size* is the size of the message buffer

The function always returns MX\_DEVICE\_OPENED.

Called By:             fl\_setup\_environment

Routines Called:     sc\_lock  
                      sc\_unlock

Parameters:           MX2\_DEVICE                   \*dev\_P  
                      INT\_4                        device\_size

Returns:             MX\_DEVICE\_OPENED

#### 2.18.34.2   mx2\_push

The mx2\_push function pushes a message onto a message buffer.

The function call is **mx2\_push(dev\_P, source\_address, message\_code, message\_size)**, where:

*dev\_P* is a pointer to the message buffer  
*source\_address* is the address of the message  
*message\_code* is the message type indicator  
*message\_size* is the size of the message in bytes

mx2\_push does the following:

- Checks to see if the buffer has room for the message.
- Calls mx2\_hwcopy to copy the message to the buffer.

The function returns MX\_MESSAGE\_PUSHED if successful. It returns MX\_DEVICE\_FULL if the message buffer is already full.

Called By:

- f0\_3dlut\_download
- f0\_3dlut\_switch
- f0\_alllut\_switch
- f0\_debug\_disable
- f0\_debug\_enable
- f0\_final\_lut\_download
- f0\_final\_lut\_switch
- f0\_mode\_select
- f0\_mpv\_lut\_type\_request
- f0\_mpv\_peek
- f0\_mpv\_poke
- f0\_mpv\_poke16
- f0\_mpv\_reset
- f0\_mpv\_write
- f0\_pass\_on
- f0\_pixel\_depth\_request
- f0\_query
- f0\_set\_display
- f0\_trigger
- gsp\_io

Routines Called:

- mx2\_hwcopy
- sc\_lock
- sc\_unlock

Parameters:	MX2_DEVICE WORD HWORD HWORD	*dev_P source_address message_code message_size
-------------	--------------------------------------	--

Returns:	MX_DEVICE_FULL MX_MESSAGE_PUSHED
----------	-------------------------------------

#### 2.18.34.3 mx2\_peek

The `mx2_peek` function previews the top message in the message buffer and returns a pointer to it.

The function call is `mx2_peek(dev_P, message_code, message_size, message_addr)`, where:

*dev\_P* is a pointer to the message buffer  
*message\_code* is a pointer to the message type indicator  
*message\_size* is a pointer to the message size in bytes  
*message\_addr* is a pointer to the message address

The function returns `MX_MESSAGE_PREVIEWED` if successful. It returns `MX_DEVICE_EMPTY` if the message buffer contains no messages.

Called By:	f1_process_messages
------------	---------------------

Routines Called:	sc_lock sc_unlock
------------------	----------------------

Parameters:	MX2_DEVICE HWORD HWORD BYTE	*dev_P *message_code *message_size **message_addr
-------------	--------------------------------------	--

Returns:	MX_DEVICE_EMPTY MX_MESSAGE_PREVIEWED
----------	---

#### 2.18.34.4 mx2\_skip

The `mx2_skip` function skips over first message in the message buffer. The skipped message is flushed from the buffer and the next message moves to the head of the queue.

The function call is `mx2_skip(dev_P)`, where *dev\_P* is a pointer to the message buffer.

The function returns `MX_MESSAGE_SKIPPED` if successful. It returns `MX_DEVICE_EMPTY` if the message buffer contains no messages.

Called By:	<code>f1_process_messages</code>	
Routines Called:	<code>sc_lock</code> <code>sc_unlock</code>	
Parameters:	<code>MX2_DEVICE</code>	<code>*dev_P</code>
Returns:	<code>MX_MESSAGE_SKIPPED</code> <code>MX_DEVICE_EMPTY</code>	

#### 2.18.34.5 `mx2_error`

The `mx2_error` function returns an ASCII translation of an error code, for output to the operator.

The function call is `mx2_error(status)`, where *status* is the error condition.

This function is not currently used.

Called By:	<code>none</code>	
Routines Called:	<code>none</code>	
Parameters:	<code>WORD</code>	<code>status</code>
Returns:	<code>"DEVICE CLOSED"</code> <code>"DEVICE TABLE FULL"</code> <code>"DEVICE OPENED"</code> <code>"DEVICE BUSY"</code> <code>"DEVICE EMPTY"</code> <code>"DEVICE FULL"</code> <code>"MESSAGE PUSHED"</code> <code>"MESSAGE POPPED"</code> <code>"MESSAGE PREVIEWED"</code> <code>"MESSAGE SKIPPED"</code> <code>"UNDEFINED ERROR"</code> <code>"UNDEFINED RETURN"</code>	

#### 2.18.34.6 `mx2_hwcopy`

The `mx2_hwcopy` function does a half-word block copy.

The function call is **mx2\_hwcopy(source\_P, destination\_P, byte\_count)**, where:

*source\_P* is the source location

*destination\_P* is the destination location

*byte\_count* is the number of bytes to be copied

Called By:           mx2\_push

Routines Called:    none

Parameters:	INT_2	*source_P
	INT_2	*destination_P
	INT_2	byte_count

Returns:            none

### 2.18.35    nmi\_type.c

The **nmi\_type** function starts the GSP task. This function is called if the command in the Force front-end control register is SUBSYS\_NMI\_START.

The function call is **nmi\_type()**. The function does the following:

- Puts the GSP start address into a data buffer.
- Calls **gsp\_ioctl\_write** to set the host interface control word to write mode.
- Calls **gsp\_write** to write the start address into the NMI vector area of GSP memory.
- Calls **gsp\_ioctl\_write** to set the GSP host interface control register to flush and clear the GSP cache.
- Calls **gsp\_ioctl\_write** to set the GSP host interface control register to start the GSP task.
- Sets the NMI flag for other routines to check before writing to the control register.
- Clears the FE\_CONTROL ready bit.

The NMI (non-maskable interrupt) is a bit in the GSP host interface control register.

Called By:           main

Routines Called:    **gsp\_ioctl\_write**  
                      **gsp\_write**

Parameters:         none

Returns:            none

### 2.18.36 poll\_ready.c

The poll\_ready function polls the ready bit in the Force front-end control register until the bit is set. (This register is used to pass messages from the real-time software to the Force task.) poll\_ready then returns the upper byte of the value in the control register. This value identifies the command to be performed.

The function call is **poll\_ready()**. The function does the following:

- Sets up the address for the front-end control register.
- Turns the Force lights on.
- Checks the control\_change variable in the Force mailbox to see if the Simulation Host has specified a new 3-D color lookup table. If yes:
  - Calls se\_clock to read and store the Force timer values.
  - Calls gsp\_ioctl\_write to set the host control word for write mode.
  - Calls gsp\_write to notify the GSP of the table switch.
  - Calls se\_clock to read and store the Force timer values.
- Checks the control\_change variable in the Force mailbox to see if the Simulation Host has specified a new 2-D (final) lookup table. If yes:
  - Calls se\_clock to read and store the Force timer values.
  - Calls gsp\_write to notify the GSP of the table switch.
  - Calls se\_clock to read and store the Force timer values.
- Checks the req\_control variable in the Force mailbox to see if the Simulation Host has specified changes to the video control registers. If yes:
  - Calls se\_clock to read and store the Force timer values.
  - Calls gsp\_ioctl\_write to set the host control word for write mode.
  - Calls gsp\_write to notify the GSP of the new control values.
  - Calls se\_clock to read and store the Force timer values.
- Turns the Force lights off.
- If the ready bit is set:
  - Turns the Force lights off.
  - Calls se\_clock to read and store the Force timer values.
  - Returns the upper byte of the front-end control register.
- Calls f1\_process\_messages to process the messages placed in the Force message buffers by the real-time software.
- Turns the Force lights off.

The function returns the upper byte of the value in the Force front-end control register.

Called By: main

Routines Called: f1\_process\_messages  
gsp\_ioctl\_write  
gsp\_write  
se\_clock

Parameters: none

Returns: (\*fe\_ctl & MSG\_MASK) >> 4

### 2.18.37 test\_gsp.c

The test\_gsp function writes a pattern to GSP memory, reads it back, and compares the data. This function is called by main if the command in the FE\_CONTROL register is SUBSYS\_TEST\_MEM.

The function call is test\_gsp(). test\_gsp does the following:

- Writes a test pattern to a buffer area.
- Calls gsp\_ioctl\_write to set the host control register for write mode.
- Calls gsp\_write to write the data in the SUB\_DATA\_BUFF buffer to GSP memory.
- Calls gsp\_ioctl\_write to set the host control register for read mode.
- Calls gsp\_read to read GSP memory into the RETURN\_DATA\_BUFF buffer.
- Compares the data in the two buffers.

The function reports the number of errors detected as *err\_count*.

Called By: main

Routines Called: gsp\_ioctl\_write  
gsp\_read  
gsp\_write

Parameters: none

Returns: err\_count

### **3 RESOURCE UTILIZATION**

#### **3.1 Disk Space Requirements**

The total amount of disk space required to house the object files for all of the CIG real-time functions is <<TBD>> bytes (approximately <<TBD>> megabytes).

#### **3.2 Memory Requirements**

The system's memory requirements vary based on application-specific parameters and system options. In general, a minimum of <<TBD>> megabytes of CPU memory is required.

A minimum of <<TBD>> megabytes of memory is required for active area memory. Additional AAM memory is required for databases with an extended viewing range (greater than 4000 meters).



## APPENDIX A. SYSTEM INCLUDE FILES

Include files define data structures and parameters used throughout the system. Although many include files are used exclusively by functions in one area, others are used by multiple CSCs. For easy reference, all include files are described in this appendix, in alphabetical order. Each subsection title identifies the full path name of the file.

### A.1 /cig/include/backend.h

The backend.h file contains the typedefs for the AAM and backend object structures. It also defines the maximum number of backends.

Included By:	aa_init.c
	backend_color.c
	backend_laser.c
	backend_man.c
	backend_thermal.c
	backend_video.c
	be_stubs.c
	cal.c
	esifa_man.c
	mpvideo_man.c
	msg_calibration_image.c

### A.2 /cig/include/bal\_real\_time.h

The bal\_real\_time.h file includes the following real-time software header files, used by Ballistics: rt\_definitions.h, rt\_types.h, and rt\_macros.h.

Included By:	b0_*.c (all files)
	bx_*.c (all files)
	gos_bal_query
	mx_*.c (all files)
	shot_report.c

### A.3 /cig/include/bal\_struct.h

The bal\_struct.h file defines the Ballistics structures for chord data, trajectory position, trajectory data, and projected data. This file also includes standard.h and sim\_cig\_if.h

Included By:	real_timevpt.h
--------------	----------------

### A.4 /cig/include/ballistics.h

The ballistics.h file includes all of the common Ballistics header files:

- bx\_defines.h
- bx\_messages.h
- bx\_rtdb\_structs.h
- bx\_structs.h

- bx\_macros.h
- bm\_functions.h
- mx\_defines.h
- slave133\_functions.h (if running on a Slave board)

Included By:        b0\_\*.c (all files)  
                     bx\_\*.c (all files)  
                     flea\_decode\_data.c  
                     gos\_bal\_query.c  
                     shot\_report.c

#### **A.5   /cig/include/bbnctype.h**

The bbnctype.h file defines character-testing macros (isalpha, isdigit, isascii, etc.) and character-conversion macros (tolower, toupper, and toascii).

Included By:        bbnctype.c

#### **A.6   /cig/include/bm\_functions.h**

The bm\_functions.h file declares all Ballistics message functions (b0\_bal\_config, b0\_database\_info, b0\_add\_traj\_table, etc.).

Included By:        ballistics.h  
                     bx\_init.c  
                     bx\_reset.c

#### **A.7   /cig/include/bp\_functions.h**

The bp\_functions.h file is not currently used.

#### **A.8   /cig/include/bx\_defines.h**

The bx\_defines.h file defines the following:

- The MALLOC macro (described in Appendix B).
- The maximum number of bvol types, model types, AAM partitions, messages, static vehicles, rounds, bvol cache entries, polygon cache entries, load modules, vehicle load modules, trajectories, terrain feedback points, and terrain feedback vehicles.
- DTP data transformation commands.
- DTP data components commands.
- DTP data traversal commands.
- Database effect model numbers.

Included By:        ballistics.h

### A.9 /cig/include/bx\_externs.h

The bx\_externs.h file declares external variables for Ballistics, including:

- Input and output buffers.
- Global (G\_\*) variables.
- Temporary variables used for message processing.

Included By:           b0\_\*.c (all files)  
                  bx\_chord\_intersect.c  
                  bx\_compute\_round.c  
                  bx\_find\_vehicle.c  
                  bx\_functions.c  
                  bx\_get\_lm\_data.c  
                  bx\_init.c  
                  bx\_model\_int.c  
                  bx\_probe.c  
                  bx\_reset.c  
                  bx\_tf\_pack.c  
                  bx\_trajectory.c  
                  gos\_bal\_query.c

### A.10 /cig/include/bx\_globals.h

The bx\_globals.h file declares variables for Ballistics, including:

- Input and output buffers.
- Global (G\_\*) variables.
- Temporary variables used for message processing.

Included By:           bx\_task

### A.11 /cig/include/bx\_macros.h

The bx\_macros.h file defines the following macros used by various functions in Ballistics:

- DELETE\_ROUND
- DELETE\_STAT\_VEH
- FREE\_LM\_CACHE
- GET\_CHORD\_END
- GET\_DB\_POS
- GET\_LB\_FROM\_LM
- NEW\_ROUND
- NEW\_STAT\_VEH

These macros are described in Appendix B.

Included By:           ballistics.h

### A.12 /cig/include/bx\_messages.h

The bx\_messages.h file contains the following:

- Declaration of the maximum message size.
- Definitions for the bal\_board\_type (Ballistics board type) variable.
- Definitions for code trace bits.
- The addresses of the boards.
- Typedefs for all simulation-to-Ballistics (MSG\_B0\_\*) messages.
- Typedefs for all Ballistics-to-simulation (MSG\_B1\_\*) messages.

Included By:

- bal\_routines.c
- ballistics.h
- config\_ballistics.c
- db\_mcc\_setup.c
- download\_bvols.c
- load\_modules.c
- msg\_process\_round48.c
- open\_dbase.c
- open\_ded.c
- rowcol\_rd.c
- simulation.c

### A.13 /cig/include/bx\_rtdb\_structs.h

The bx\_rtdb\_structs.h file defines the structure of the real-time database for Ballistics. It includes typedefs for the following:

- Floating bounding volume entry.
- Single-transform model structure.
- Show effects stamp structure.
- Tank structure.
- Database directory entry.
- Runtime database header.
- Fixed bounding volume entry.
- Generic load module directory entry.
- Grid components.
- Grid locator information.
- Load module header.
- Load module statistics (generic model, unique static, and terrain grid polygon count, plus total bytes per load module).
- Polygon data (info word).
- Polygon list of vertices and alpha betas for *texturing*.

This file also defines the maximum number of models that can be put in the generic module of the runtime database, the maximum number of stamps possible in one unique static object definition, and the number of z values in a grid component.

Included By: ballistics.h

#### A.14 /cig/include/bx\_structs.h

The bx\_structs.h file contains structure definitions for Ballistics. It includes typedefs for the following:

- Load module/grid search list.
- Static vehicle.
- Bounding volume cache entry.
- Terrain and object polygon.
- Polygon cache entry.
- Load module cache entry.
- Trajectory table entry.
- Trajectory table.
- Point data.
- Chord.
- Round data.
- Terrain corners.

Included By:           ballistics.h

#### A.15 /cig/include/cigsimio.h

The cigsimio.h file contains the structures used for writing messages to a file during runtime (to record a script for later playback or to display messages to stdout). This file defines the message structure and the tag records.

Included By:           cigsimio\_obj.c  
                  flea\_script.c  
                  host\_if\_debug.c  
                  print\_msg.c

#### A.16 /cig/include/clouds.h

The clouds.h file defines cloud control structures, including cloud states and cloud control blocks.

Included By:           flea.c  
                  gossip.c  
                  init\_sim.c  
                  msg\_cig\_ctl.c  
                  msg\_drll.c  
                  msg\_end.c  
                  msg\_pass\_on.c  
                  msg\_subsys\_mode.c  
                  msg\_vport.c  
                  real\_time.h  
                  real\_timevpt.h  
                  simulation.c

### A.17 /gt/include/clparse.h

The clparse.h file is used to parse the command line entered to start up the system. It defines the structure used to define each valid switch.

Included By:       bx147\_main.c  
                    loadmpv.c  
                    rtt.c  
                    rtt\_init.c

### A.18 /cig/include/db\_struct.h

The db\_struct.h file defines MAX\_DB\_PARTITIONS and GMAL\_SIZE. It also contains the typedefs for the database\_pointers, db\_info, and lms\_data (load module specific data) structures.

This file includes standard.h and model\_struct.h.

Included By:       real\_timevpt.h

### A.19 /cig/include/def\_alloc.h

The def\_alloc.h file defines DYNAMIC as 0 and SYSTEM as 1.

Included By:       cig\_config.c  
                    clouds.c  
                    flagoff.c  
                    gun\_overlays.c  
                    init\_free.c  
                    msg\_end.c  
                    mtx\_concat.c  
                    path.c  
                    path\_init.c  
                    real\_time.h  
                    vpt\_set.c

### A.20 /cig/include/def\_mtx\_type.h

The def\_mtx\_type.h file defines configuration node types (MATRIX\_NODE, BRANCH\_NODE, and BRANCH\_MTX\_NODE) and matrix types (RTS3x3\_TYPE, RTS4x3\_TYPE, ROT2x1\_TYPE, TRANSLATION, SCALE, ROT1\_TYPE, and ROT3\_TYPE).

Included By:       encode\_routines.c  
                    flea\_bal\_opts.c  
                    flea\_init\_cig\_sw.c  
                    print\_msg.c  
                    tick.c  
                    vpi\_struct.h

### A.21 /cig/include/defines\_2d.h

The defines\_2d.h file contains definitions used by the 2-D compiler, including:

- All 2-D database commands (N\_\*, A\_\*, and B\_\*).
- Return codes (end of file, too many errors, invalid window number, etc.).
- Color, plane, and static/dynamic commands.
- MPV addresses (base component pointers and base program area).
- MPV default screen parameters (e.g., dimensions and pitch).
- MPV space allocation.
- Array sizes (maximum number of component pointers, windows, component descriptions, etc.).
- The maximum number of 2-D compiler errors allowed before the compilation is aborted.

Included By:        global\_2d.h  
                     globfir\_2d.h

### A.22 /cig/include/definitions.h

The definitions.h file defines miscellaneous constants and structures used by the real-time software. It includes:

- Various definitions used for by Ballistics to parse bounding volume structures and report hits.
- Definitions of various macros (ABSVAL, SET\_OUT\_BITS, SET\_OUT\_M2BITS, XREAD, XOPEN, XCLOSE, XLSEEK, XWRITE, AAREAD). These are described in Appendix B.
- The typedef for the load module/grid search list structure.
- Pointers for messages and other parameters.

Included By:        flea.c  
                     flea\_bal\_opts.c  
                     flea\_simulate\_vehicles.c  
                     global\_init.c  
                     gossip.c  
                     host\_dr11\_if.c  
                     host\_mpv\_if.c  
                     host\_socket\_if.c  
                     init\_sim.c  
                     mpvideo\_response.c  
                     msg\_cig\_ctl.c  
                     msg\_dr11.c  
                     msg\_end.c  
                     msg\_laser.c  
                     msg\_lt\_state.c  
                     msg\_pass\_on.c  
                     msg\_ppm.c  
                     msg\_process\_round48.c  
                     msg\_subsys\_mode.c  
                     real\_time.h

real\_timevpt.h  
msg\_vport.c  
rtt\_init.c  
simulation.c

### A.23 /cig/include/demo\_struct.h

The demo\_struct.h file specifies the following:

- Various limits for Flea exercises (MAX\_DEMO\_MODELS, MAX\_HITS, MAX\_COLLISIONS, etc.).
- The typedef for the DEMO1 structure used for Flea demos (model rotation, starting position, spacing between models, etc.).
- The typedefs for the F\_SIMIN and F\_SIMOUT structures, used to exchange messages with the real-time software during a Flea exercise.
- The typedef for the POI (point of interest) structure.
- The typedef for the SIM\_VEH\_STRUCT (simulated vehicle) structure.

Included By:

dynamic\_demo.c  
encode\_routines.c  
flea.c  
flea\_agl\_terrain\_follow.c  
flea\_agpt\_locations.c  
flea\_agpt\_switches.c  
flea\_bal\_opts.c  
flea\_db\_traverse.c  
flea\_decode\_data.c  
flea\_encode\_data.c  
flea\_init\_cig\_sw.c  
flea\_ppm\_obj.c  
flea\_script.c  
flea\_simulate\_vehicles.c  
flea\_switches.c  
flea\_update\_pos.c  
flea\_veh\_control.c  
get\_sio\_data.c  
model\_demo.c  
tick.c  
tick\_ppm.c  
tick\_script.c  
update\_2d.c  
update\_agpt\_2d.c

### A.24 /cig/include/dgi\_std.h

The dgi\_std.h file helps make the code compiler-independent (avoiding differences in storage allocation caused by differences in C compilers) by defining basic data types, as follows:



Storage Allocated	C Language	Independent Data Type
18 bit char	char	char
16 bit signed integer	short	INT_2
32 bit signed integer	int	INT_4
8 bit unsigned integer	unsigned char	BYTE, BOOLEAN
16 bit unsigned integer	unsigned short	HWORD
32 bit unsigned integer	unsigned int	WORD
32 bit real	float	REAL_4
64 bit real	double	REAL_8

This file is also located in the /cig/othersrc/force directory.

Included By:

- aa\_init.c
- backend\_\*.c (all files)
- bit\_blt.c
- bootforce.c
- bootmpv.c
- cf\_translator.c
- cig\_2d\_setup.c
- cig\_comp\_2d.c
- comp.c
- draw\_line.c
- esifa\_\*.c (all files)
- f0\_\*.c (all files)
- f1\_\*.c (all files)
- flea.c
- flea\_bal\_opts.c
- flea\_simulate\_vehicles.c
- forcetask.c
- get\_thing.c
- get\_vehicle\_position.c
- global\_init.c
- gossip.c
- gsp\_io.c
- host\_dr11\_if.c
- host\_ex\_if.c
- host\_mpv\_if.c
- host\_socket\_if.c
- init\_sim.c
- init\_stuff.c
- load\_esifa.c
- loadmpv.c
- mpvideo\_\*.c (all files)
- msg\_\*.c (all files except msg\_calibration\_image.c)
- mx2\_hword.c
- mx3\_hword.c
- nmi\_type.c
- otherveh\_state.c
- oval\_rect.c
- poll\_ready.c

```
poly.c
ppm_obj.c
pretend_veh.c
print_msg.c
proc_cmd.c
real_time.h
replace_mod.c
simulation.c
staticveh_remove.c
staticveh_state.c
string.c
sys_control.c
sysdefs2.h
test_commands.c
test_gsp.c
text.c
u_*_2d.c (all files)
window.c
```

### A.25 /cig/include/dgi\_stdg.h

The dgi\_stdg.h file defines various graphics structures. It includes typedefs for the following:

- 2-D, 3-D, and 4-D vertex points.
- 4x3 matrix.
- 2-D and 3-D bounding boxes.
- Red, green, blue.
- Red, green, blue, opaque.
- Hue, saturation, lightness.
- Hue, saturation, lightness, opaque.

Included By:

```
cf_translator.c
flea.c
flea_bal_opts.c
flea_simulate_vehicles.c
get_vehicle_position.c
global_init.c
gossip.c
host_dr11_if.c
host_ex_if.c
host_mpv_if.c
host_socket_if.c
init_sim.c
mpvideo_response.c
msg_*.c (all files except msg_calibration_image.c and
        msg_syserr.c)
otherveh_state.c
pretend_veh.c
print_msg.c
real_time.h
simulation.c
staticveh_remove.c
```

staticveh\_state.c

#### A.26 /cig/include/dig\_defines.h

The dig\_defines.h file defines variables used for the digitizer interface.

Included By:        flea\_demo.c  
                     gos\_locate.c

#### A.27 /cig/include/dig\_struct.h

The dig\_struct.h file defines the DIGITIZER\_INTERFACE structure.

Included By:        flea\_demo.c

#### A.28 /cig/include/ecompile1.h

The ecompiler1.h file contains defines for the DTP command generator, including various DTP addresses, maximum values, and the typedef for the where\_process structure (used for pre- and post-processing models and effects).

This file includes the real\_time.h and ememory\_map.h files.

Included By:        dtp\_compiler.c  
                     dtp\_funcs.c  
                     dtp\_trav1.c  
                     dtp\_trav2.c  
                     gossip.c  
                     init\_sim.c  
                     msg\_cig\_ctl.c  
                     msg\_dr11.c  
                     msg\_end.c  
                     msg\_pass\_on.c  
                     msg\_vport.c  
                     otherveh\_state.c  
                     real\_time.h  
                     real\_timevpt.h  
                     simulation.c  
                     staticveh\_remove.c  
                     staticveh\_state.c

#### A.29 /cig/include/ememory\_map.h

The ememory\_map.h file provides external memory declarations. It includes the following:

- General-use variables, such as My Vehicle id, names of the loaded files, and the database column markers.
- Database format variables.
- Database management variables, such as the number of load modules on a side, load module width, and the total number of load modules.

- Variables for ballistics and flea.
- Timing and control word variables.
- Local terrain and range variables.
- Declarations for the DR11-W interface.
- Declarations to support runtime configurable DR packet sizes.
- Intertask Semaphore mailbox declarations.
- Debugging and data gathering variables.
- Variables for Flea's keyboard interface.
- FOGM missile mode global variables.
- Variables used with Force and GSP.
- Single step flags.
- Ballistics flags.
- Helicopter blade rotation variables.
- The GLOB (global memory) macro.

This file also includes the memory\_map\_defines.h file.

Included By:

- aam\_manager.c
- autopilot.c
- bal\_routines.c
- be\_stubs.c
- bx\_task.c
- cal.c
- cig\_config.c
- cig\_getm\_2d.c
- cigsimio\_obj.c
- close\_db.c
- clouds.c
- config\_ballistics.c
- config\_color\_table.c
- config\_database.c
- db\_mcc\_setup.c
- debug\_initdr.c
- ded\_model\_trace.c
- dtp\_emu.c
- dynamic\_demo.c
- effect\_downcount.c
- file\_control.c
- flea.c
- flea\_\*.c (all files except flea\_simulate\_vehicles.c)
- generic\_lm.c
- get\_sio\_data.c
- get\_tx\_lut\_index.c
- gos\_120tx.c
- gos\_db\_query.c
- gos\_memory.c
- gos\_model.c
- gos\_mpv.c
- gos\_mpvio.c
- gos\_polys.c
- gos\_system.c
- gossip.c
- gun\_overlays.c
- host\_enet\_if.c

host\_flea\_if.c  
host\_if\_debug.c  
host\_scsi\_if.c  
hw\_test.c  
load\_dbase.c  
load\_modules.c  
loc\_ter.c  
loc\_ter\_msg.c  
make\_bbn.c  
make\_bbn\_logo.c  
mkcal.c  
model\_demo.c  
model\_mtx.c  
model\_mtx.c  
msg\_calibration\_image.c  
msg\_cig\_ctl.c  
msg\_drll.c  
msg\_end.c  
msg\_pass\_on.c  
msg\_subsys\_mode.c  
msg\_vport.c  
open\_dbase.c  
open\_ded.c  
rowcol\_rd.c  
show\_effect\_msg.c  
simulation.c  
tick.c  
tick\_ppm.c  
tick\_script.c  
update\_2d.c  
update\_agpt\_2d.c

### A.30 /cig/include/esifa.h

The esifa.h file defines the structures used to communicate with the ESIFA board on the SCSI bus using the ifx\_ioctl call. It includes the typedefs for the ESIFA object and the ESIFA ports structure.

Included By:

- backend\_color.c
- backend\_man.c
- esifa\_\*.c (all files)
- flea.c
- gossip.c
- msg\_cig\_ctl.c
- msg\_drll.c
- msg\_end.c
- msg\_pass\_on.c
- msg\_subsys\_mode.c
- msg\_vport.c
- ppm\_obj.c
- real\_time.h
- real\_timevpt.h
- simulation.c

### A.31 /cig/othersrc/force/f1\_defines.h

The f1\_defines.h file defines Force constants such as the following:

- The size of each Force message buffer.
- The base address of each Force message buffer.
- Typedefs for the MODE\_SELECT\_STRUCT and SWITCH\_2DLUT\_STRUCT structures.

Included By:        f0\_\*.c (all files)  
                      f1\_\*.c (all files)  
                      f1\_defines.h  
                      forcetask.c  
                      gsp\_io.c  
                      poll\_ready.c

### A.32 /cig/othersrc/force/f1\_externs.h

The f1\_externs.h file defines various external Force global variables, including:

- The pointer to the Force environment.
- Pointers to the message buffers.
- nmi\_set\_flag (indicates whether or not GSP is running).
- Force scratch buffer (force\_tmp\_buffer[]).
- Function jump table for message processing (\*FG\_message\_table[]).
- Local analogues of the MPV registers.
- FG\_mpv\_type, FG\_final\_lut\_type, FG\_mpv\_mode, and FG\_mpv\_orient.
- FG\_pixel\_addr[], FG\_pixel\_id[], and FG\_new\_pixel\_req\_flag[].
- FG\_mpv\_cluts\_gsp and FG\_mpv\_cluts\_force.
- FG\_3dlut\_downloaded\_flag.
- FG\_pixel\_depth\_message and FG\_pixel\_depth\_flag[].
- FG\_debug\_flag.

Included By:        f0\_\*.c (all files)  
                      f1\_externs.h  
                      f1\_force\_init.c  
                      f1\_init\_jump\_table.c  
                      f1\_process\_messages.c  
                      f1\_setup\_environment.c  
                      gsp\_io.c  
                      poll\_ready.c

### A.33 /cig/othersrc/force/f1\_globals.h

The f1\_globals.h file defines various Force global variables, including:

- Pointers to the message buffers.
- nmi\_set\_flag (indicates whether or not GSP is running).
- Force scratch buffer (force\_tmp\_buffer[]).
- \*force\_interface (mailbox address)

- Function jump table for message processing (\*FG\_message\_table[]).
- Local analogues of the MPV registers.
- FG\_mpv\_type, FG\_final\_lut\_type, FG\_mpv\_mode, and FG\_mpv\_orient.
- FG\_pixel\_addr[], FG\_pixel\_id[], and FG\_new\_pixel\_req\_flag[].
- FG\_mpv\_cluts\_gsp and FG\_mpv\_cluts\_force.
- FG\_3dlut\_downloaded\_flag.
- FG\_pixel\_depth\_message and FG\_pixel\_depth\_flag[].
- FG\_debug\_flag.

Included By:           forcetask.c

#### A.34 /cig/othersrc/force/force\_defines.h

The force\_defines.h file, which contains Force and GSP definitions, serves as the interface between the real-time software, Force, and the GSP. It includes defines for the following:

- FE\_CONTROL (the front-end control register).
- FORCE\_CONTROL (the Force control register).
- Force return status and error areas.
- Pixel depth request values.
- Lookup table variables.
- Incoming and outgoing data buffer areas.
- Video control variables.
- GSP host control interface words.
- The READ\_CLOCK, RESTART\_CLOCK, and CHECK\_CLOCK macros (no longer used).
- The typedef for the FORCE\_INTERFACE structure.

Included By:           data\_type.c  
                   f0\_set\_display.c  
                   gsp\_io.c  
                   nmi\_type.c  
                   test\_gsp.c

#### A.35 /cig/othersrc/force/force\_env.h

The force\_env.h file contains the typedef for the FORCE\_ENVIRONMENT structure, which defines Force memory.

This file is also located in the /cig/include directory.

Included By:           f1\_defines.h  
                   f1\_init\_jump\_table.c  
                   f1\_process\_messages.c  
                   force\_env.h  
                   force\_rtsw\_if.h

#### A.36 /cig/othersrc/force/force\_rtsw\_if.h

The force\_rtsw\_if.h file defines the interface between Force and the real-time software. It defines the following:

- MAX\_ORIENTATIONS and MAX\_RESOLUTIONS.
- ORIENT\_VERTICAL and ORIENT\_HORIZONTAL.
- RES\_640x180, RES\_320x240, RES\_640x256, and RES\_640x240.
- The FORCE\_ENVIRONMENT structure.
- The message code and body of the MSG\_F0\_QUERY message.
- The message code and body of the MSG\_F1\_TEXT message.

Included By: none

### A.37 /cig/othersrc/force/force.h.asm

The force.h.asm file defines constants for the Force data link. It sets up the 68230 base register and defines GCR codes, address select codes for GSP registers, and LED bit definitions.

Included By: force.asm

### A.38 /cig/include/functions.h

The functions.h file defines the following macros used by various functions in the real-time software. These macros are described in Appendix B.

- AAM2\_ADDR
- DART\_ENQUEUE
- DUMP\_DART\_BUFFER
- EXCHANGE\_FLEA\_DATA
- FIND\_LM
- FXTO881
- INIT\_MTX
- OPEN\_FLEA\_DATA
- SYSERR
- TRIGGER\_FORCE
- WAIT\_FORCE

Included By:

- backend\_color.c
- backend\_paths.c
- backend\_thermal.c
- cf\_translator.c
- esifa\_laser.c
- esifa\_man.c
- flea.c
- flea\_simulate\_vehicles.c
- init\_sim.c
- mpvideo\_laser.c
- mpvideo\_man.c
- mpvideo\_response.c
- msg\_cig\_ctl.c
- msg\_drll.c
- msg\_end.c
- msg\_laser.c
- msg\_laser\_return.c



msg\_lt\_state.c  
msg\_pass\_on.c  
msg\_ppm.c  
msg\_process\_round48.c  
msg\_vport.c  
otherveh\_state.c  
real\_time.h  
real\_timevpt.h  
staticveh\_remove.c  
staticveh\_state.c

#### A.39 /cig/include/global\_2d.h

The global\_2d.h file includes the defines\_2d.h and struct\_2d.h include files. Collectively, these files declare all global I/O variables, global temporary compiler variables, and compiler product variables for the 2-D compiler.

Included By:

bit\_blt.c  
cig\_comp\_2d.c  
cig\_getm\_2d.c  
cig\_link\_2d.c  
comp.c  
draw\_line.c  
get\_thing.c  
init\_stuff.c  
oval\_rect.c  
poly.c  
proc\_cmd.c  
string.c  
text.c  
u\_comp\_2d.c  
u\_getm\_2d.c  
u\_link\_2d.c  
window.c

#### A.40 /cig/include/globfir\_2d.h

The globfir\_2d.h file includes the defines\_2d.h and struct\_2d.h include files. Collectively, these files declare all global I/O variables, global temporary compiler variables, and compiler product variables for the 2-D compiler.

Included By:

cig\_2d\_setup.c  
u\_main2d.c

#### A.41 /cig/include/if\_cig2sim.h

The if\_cig2sim.h file defines the structures of the runtime messages sent from the CIG to the Simulation Host. These messages are:

- MSG\_AGL
- MSG\_HIT\_RETURN

- MSG\_LASER\_RETURN
- MSG\_LOCAL\_TERRAIN
- MSG\_LT\_PIECE
- MSG\_MISS
- MSG\_PASS\_BACK
- MSG\_RETURN\_POINT\_INFO
- MSG\_SHOT\_REPORT
- MSG\_TF\_HDR
- MSG\_TF\_PT

Included By:

- flea.c
- flea\_bal\_opts.c
- flea\_simulate\_vehicles.c
- global\_init.c
- gossip.c
- init\_sim.c
- msg\_cig\_ctl.c
- msg\_dr11.c
- msg\_effect.c
- msg\_end.c
- msg\_laser\_return.c
- msg\_pass\_back.c
- msg\_pass\_on.c
- msg\_process\_round48.c
- msg\_subsys\_mode.c
- msg\_veh\_state.c
- msg\_vport.c
- print\_msg.c
- sim\_cig\_if.h

#### A.42 /cig/include/if\_ctl\_err.h

The if\_ctl\_err.h file defines the structures of the control and error messages passed between the CIG and the Simulation Host. These messages are:

- MSG\_CIG\_CTL
- MSG\_DR11\_PKT\_SIZE
- MSG\_SYS\_ERROR

This file also defines the structure of the message header (MSG\_HDR), end of message indicator (MSG\_END), and packet header (MSG\_BLK).

Included By:

- flea.c
- flea\_bal\_opts.c
- flea\_simulate\_vehicles.c
- global\_init.c
- gossip.c
- host\_dr11\_if.c
- host\_ex\_if.c
- host\_mpv\_if.c
- host\_socket\_if.c
- init\_sim.c
- mpvideo\_response.c

msg\_\*.c (all files except msg\_calibration\_image.c)  
otherveh\_state.c  
pretend\_veh.c  
print\_msg.c  
sim\_cig\_if.h  
simulation.c  
staticveh\_remove.c  
staticveh\_state.c

#### A.43 /cig/include/if\_hdr\_str.h

The if\_hdr\_str.h defines the matrix structures (RTS4x3\_MTX, RTS3x3\_MTX, ROT2x1\_MTX, MTXUNION, and XYZHPR). It also defines the structures for field-of-view, resolution, and the vector for describing angles of rotation.

Included By:

- flea.c
- flea\_bal\_opts.c
- flea\_simulate\_vehicles.c
- global\_init.c
- gossip.c
- init\_sim.c
- mpvideo\_response.c
- msg\_\*.c (all files except msg\_calibration\_image.c and msg\_syserr.c)
- otherveh\_state.c
- print\_msg.c
- sim\_cig\_if.h
- simulation.c
- staticveh\_remove.c
- staticveh\_state.c

#### A.44 /cig/include/if\_init.h

The if\_init.h file defines the structures of the initialization messages sent to the CIG from the Simulation Host. These messages are:

- MSG\_2D\_SETUP
- MSG\_TRAJ\_TABLE\_XFER
- MSG\_TRAJ\_ENTRY\_XFER
- MSG\_CREATE\_CONFIGNODE
- MSG\_VIEWPORT\_STATE
- MSG\_OVERLAY\_SETUP
- MSG\_DEFINE\_TX\_MODE
- MSG\_AMMO\_DEFINE

Included By:

- gossip.c
- print\_msg.c
- sim\_cig\_if.h
- simulation.c

**A.45 /cig/include/if\_msg\_ids.h**

The if\_msg\_ids.h file defines the numerical message code for each valid message type.

Included By:        flea.c  
                     gossip.c  
                     host\_ex\_if.c  
                     msg\_cig\_ctl.c  
                     msg\_dr11.c  
                     msg\_end.c  
                     msg\_pass\_on.c  
                     msg\_subsys\_mode.c  
                     msg\_vport.c  
                     simulation.c

**A.46 /cig/include/if\_rva2net.h**

The if\_rva2net.h file provides the message structures for alternate versions of some of the standard Ballistics messages. These messages, which allow for 48-bit identifiers, are not used with the standard GT100 system.

- MSG\_HIT\_RETURN48
- MSG\_TF\_PT48
- MSG\_SHOT\_REPORT48

Included By:        msg\_process\_round48.c  
                     print\_msg.c

**A.47 /cig/include/if\_sim2cig.h**

The if\_sim2cig.h file defines the structures of the messages sent to the CIG by the Simulation Host. These messages are:

- MSG\_1ROTATION
- MSG\_3ROTATIONS
- MSG\_ADD\_TRAJ\_TABLE
- MSG\_AGL\_SETUP
- MSG\_CALIBRATION\_IMAGE
- MSG\_CANCEL\_ROUND
- MSG\_CLOUD\_STATE
- MSG\_DATA\_TEST
- MSG\_DELETE\_TRAJ\_TABLE
- MSG\_GUN\_OVERLAY
- MSG\_HPRXYZS\_MATRIX
- MSG\_LT\_STATE
- MSG\_OBSCURE
- MSG\_OTHERVEH\_STATE
- MSG\_PASS\_ON
- MSG\_PPM\_DISPLAY\_MODE
- MSG\_PPM\_DISPLAY\_OFFSET

- MSG\_PPM\_PIXEL\_LOCATION
- MSG\_PPM\_PIXEL\_STATE
- MSG\_PROCESS\_CHORD
- MSG\_PROCESS\_ROUND
- MSG\_PROCESS\_ROUND48
- MSG\_REQUEST\_LASER\_RANGE
- MSG\_REQUEST\_POINT\_INFO
- MSG\_ROT2x1\_MATRIX
- MSG\_ROUND\_FIRED
- MSG\_RTS4x3\_MATRIX
- MSG\_SCALE
- MSG\_SHOW\_EFFECT
- MSG\_SIO\_CLOSE
- MSG\_SIO\_INIT
- MSG\_SIO\_WRITE
- MSG\_STATICVEH\_REM
- MSG\_STATICVEH\_STATE
- MSG\_SUBSYS\_MODE
- MSG\_TF\_INIT\_HDR
- MSG\_TF\_INIT\_PT
- MSG\_TF\_STATE
- MSG\_TF\_VEHICLE\_POS
- MSG\_TRAJ\_CHORD
- MSG\_TRAJ\_ENTRY
- MSG\_TRANSLATION
- MSG\_VIEW\_FLAGS
- MSG\_VIEW\_MAGNIFICATION
- MSG\_VIEWPORT\_UPDATE

Included By:

- flea.c
- flea\_bal\_opts.c
- flea\_simulate\_vehicles.c
- gossip.c
- init\_sim.c
- msg\_cig\_ctl.c
- msg\_dr11.c
- msg\_effect.c
- msg\_end.c
- msg\_laser.c
- msg\_lt\_state.c
- msg\_pass\_on.c
- msg\_ppm.c
- msg\_process\_round48.c
- msg\_subsys\_mode.c
- msg\_veh\_state.c
- msg\_vflags.c
- msg\_vport.c
- otherveh\_state.c
- print\_msg.c
- sim\_cig\_if.h
- simulation.c
- staticveh\_remove.c
- staticveh\_state.c

**A.48 /cig/include/if\_tst\_ctl.h**

The if\_tst\_ctl.h file defines the structures of the SIM-to-CIG messages used for file control and hardware testing. These messages are:

- MSG\_FILE\_DESCR
- MSG\_FILE\_STATUS
- MSG\_FILE\_XFER
- MSG\_TEST\_NAME

Included By:       gossip.c  
                    print\_msg.c  
                    simulation.c

**A.49 /cig/include/if\_veh\_eff.h**

The if\_veh\_eff.h file contains defines that affect dynamic vehicles and special effects. It defines system limits (MAX\_NUM\_TANKS, MAX\_NUM\_VEH, MAX\_ACT\_EFF, MAX\_STAT\_TANKS, and MAX\_STAT\_VEH), ASID bumper number characters, and ASID special effects.

Included By:       gossip.c  
                    init\_sim.c  
                    msg\_cig\_ctl.c  
                    msg\_drll.c  
                    msg\_effect.c  
                    msg\_end.c  
                    msg\_pass\_on.c  
                    msg\_veh\_state.c  
                    msg\_vport.c  
                    otherveh\_state.c  
                    print\_msg.c  
                    sim\_cig\_if.h  
                    simulation.c  
                    staticveh\_remove.c  
                    staticveh\_state.c

**A.50 /gt/include/ifxvisi.h**

The ifxvisi.h file provides definitions used when using the Ready Systems IFX routines to manage and access files. It includes IFX function codes; the access modes for ifx\_open; the options for ifx\_mount, ifx\_delete, and ifx\_rmdir; I/O control operation codes; device types; IFX error codes; and the structure of the directory entry.

Included By:       bx147\_main.c  
                    cigsimio\_obj.c  
                    directory.c  
                    esifa\_load.c  
                    esifa\_man.c  
                    esifa\_query.c

```
find_gtfnc.c
flea_script.c
host_enet_if.c
host_scsi_if.c
ppm_obj.c
rtt.c
rtt_init.c
sio.c
stdopen.c
tick_script.c
```

#### **A.51 /cig/include/keywords.h**

The keywords.h file defines the table of keywords recognized by lexical analyzer in the Token Processing CSC. This file includes lex.h.

Included By:       lex.c

#### **A.52 /cig/include/kludge.h**

The kludge.h file defines the AAM1\_TO\_AAM2 and VME\_TO\_VMX macros (described in Appendix B). It also declares several of the Message Processing functions.

Included By:       msg\_cig\_ctl.c  
                  msg\_drll.c  
                  msg\_effect.c  
                  msg\_end.c  
                  msg\_veh\_state.c  
                  msg\_vport.c  
                  simulation.c

#### **A.53 /cig/include/lex.h**

The lex.h file defines the lexical tokens recognized by the lexical analyser in the Token Processing CSC.

Included By:       keyword.h  
                  lex.c  
                  subsys\_cfg\_parse.c

#### **A.54 /cig/include/m2\_config.h**

The m2\_config.h file contains defines specific to the M2. It defines channel and gunner resolution, viewport angular offsets, pitch up/down angular offsets, field-of-view sizes for all channels, and texture map definitions.

Included By:       gun\_overlays.c

**A.55 /gt/include/math.h**

The math.h file declares standard math functions.

Included By:	bal_routines.c
	bx_bvol_int.c
	bx_poly_int.c
	bx_task.c
	cnode_child.c
	cnode_process.c
	cnode_query.c
	cnode_set.c
	dtp_emu.c
	dynamic_demo.c
	encode_routines.c
	fill_tree.c
	flea_agl_terrain_follow.c
	flea_agpt_locations.c
	flea_agpt_switches.c
	flea_bal_opts.c
	flea_db_traverse.c
	flea_decode_data.c
	flea_init_cig_sw.c
	flea_ppm_obj.c
	flea_script.c
	flea_simulate_vehicles.c
	flea_switches.c
	flea_update_pos.c
	flea_veh_control.c
	gos_120tx.c
	gos_model.c
	gos_mpv.c
	gos_mpvio.c
	gossip.c
	linkvpt.c
	mkcal.c
	model_demo.c
	mtx_concat.c
	mtx_viewspace.c
	mx_error.c
	path.c
	path_init.c
	path_query.c
	shot_report.c
	simulation.c
	tick_ppm.c
	tick_script.c
	trav_tree.c
	tst_eupdate.c
	u_brmask.c
	u_path.c
	u_rotations.c



```
u_viewport.c
u_xfrm.c
update_2d.c
update_agpt_2d.c
update_mtx.c
update_rot.c
vpt_process.c
vpt_query.c
vpt_set.c
vpt_update.c
```

### A.56 /cig/include/mbx.h

The mbx.h file contains defines for the interface with the Force board. It includes defines for the following:

- The FORCE\_INTERFACE structure.
- NMI and GSP configuration addresses.
- Force masks (slave/host, resolution, front ready, force busy, etc.).
- FE\_CONTROL register commands (SUBSYS\_READ\_START, SUBSYS\_TEST\_MEM, etc.).
- Video control parameters (VIDEO\_CTL\_ADDR, VIDEO\_ON\_CODE, etc.).
- GSP memory start and end addresses.

This file is also located in the /cig/other/src/force directory.

Included By:

```
f0_*.c (all files)
f1_*.c (all files)
flea.c
forcetask.c
gossip.c
msg_cig_ctl.c
msg_dr11.c
msg_end.c
msg_pass_on.c
msg_subsys_mode.c
msg_vport.c
poll_ready.c
process_vflags.c
real_time.h
real_timevpt.h
simulation.c
```

### A.57 /cig/include/memory\_map.h

The memory\_map.h file contains external memory declarations such as the following:

- Variables describing the simulated vehicle (myveh\_id, myveh\_type, etc.).
- Database management variables.
- Local terrain and range variables.
- Terrain feedback variables.
- Declarations for the DR11-W interface.

- Intertask semaphore mailbox declarations.
- The display\_lights array.
- Viewport position and rotation data for flying and setting individual views.
- Variables used by Flea's keyboard interface for flying.
- Helicopter blade rotation variables.
- Initialized values for the message packet size and local terrain chunk size.
- The GLOB (global memory) macro, defined in Appendix B.

This file include memory\_map\_defines.h

Included By:           upstart.c

#### A.58 /cig/include/memory\_map\_defines.h

The memory\_map\_defines.h file defines variables used in external memory declarations, including the following:

- The size of a load module.
- The areas of the 64KW memory board (32KW of space for the double-buffer state table and 32KW of generic memory for the database).
- Byte offsets to data in double-buffered state table memory.
- Declarations for the DR11-W interface.
- Local terrain message interval and starting frame number.
- Intertask semaphore mailbox locations.
- Display lights locations.
- Viewport position and rotation data for flying and setting individual views.
- Helicopter blade rotation variables, used in simulation.

Included By:

- aa\_init.c
- backend\_branch.c
- backend\_color.c
- backend\_man.c
- backend\_paths.c
- backend\_thermal.c
- ball\_effect\_add.c
- cig\_link\_2d.c
- dl\_man.c
- ememory\_map.h
- esifa\_man.c
- global\_init.c
- host\_ex\_if.c
- init\_sim.c
- memory\_map.h
- mpvideo\_man.c
- msg\_effect.c
- msg\_laser\_return.c
- msg\_pass\_back.c
- msg\_veh\_state.c
- otherveh\_state.c
- process\_vflags.c
- process\_vppos.c
- rtt.c
- rtt\_init.c

staticveh\_remove.c  
staticveh\_state.c  
sys\_control.c

#### **A.59 /cig/include/model\_struct.h**

The model\_struct.h file provides the typedefs for the following model structures:

- Static tank.
- Single-transform model.
- ASID single-transform model.
- Remove static model.
- Tank.
- Show effects stamp.
- ASID show effects stamp.
- Fake dynamic vehicle.

This file includes standard.h, poly\_struct.h, and sim\_cig\_if.h.

Included By:        db\_struct.h  
                      real\_timevpt.h

#### **A.60 /cig/other/src/force/mpv\_mdef.h**

The mpv\_mdef.h file defines memory locations in the MPV that are used by Force and the 2-D task. It includes MPV register locations and locations used in the Force/MPV interface.

Included By:        fl\_defines.h

#### **A.61 /cig/other/src/force/mpv\_memory\_defines.h**

The mpv\_memory\_defines.h file defines MPV register locations.

Included By:        mpv\_mdef.h  
                      mpv\_memory\_defines.h

#### **A.62 /cig/include/mpv\_struct.h**

The mpv\_struct.h file provides typedefs for the following MPV structures:

- Color table.
- Fade transfer.
- Video control.
- DTP mode switch.

This file includes standard.h.

Included By:        process\_vflags.c  
                      real\_timevpt.h

### A.63 /cig/include/mpvideo.h

The mpvideo.h file defines the interface to the MPV board. It defines the following:

- The MPVIO\_INTERFACE structure.
- All Force commands (SUBSYS\_READ\_START, SUBSYS\_MAIL\_SEND, etc.).
- Video control commands.
- Force/GSP masks.
- The WAIT\_MPVIO macro.
- The MPVIDEO\_CONTROL and MPVIDEO\_OBJ structures.

Included By:            backend\_man.c  
                         bootforce.c  
                         bootmpv.c  
                         cig\_2d\_setup.c  
                         cig\_comp\_2d.c  
                         cig\_link\_2d.c  
                         esifa\_man.c  
                         loadmpv.c  
                         mpvideo\_\*.c (all files except mpvideo\_print.c)

### A.64 /cig/include/mpvideo\_msg.h

The mpvideo\_msg.h file provides the typedefs for all messages sent between the real-time software and the MPV via the Force board. Both M\_F0\_\* (RTSW->Force) and M\_F1\_\* (Force->RTSW) messages are included. This file also defines the message codes and various variables used in the messages.

This file is also located in the /cig/other/src/force directory.

Included By:            bootmpv.c  
                         f0\_\*.c (all files)  
                         f1\_\*.c (all files)  
                         forcetask.c  
                         gos\_mpvio.c  
                         gsp\_io.c  
                         loadmpv.c  
                         mpvideo\_laser.c  
                         mpvideo\_lut.c  
                         mpvideo\_man.c  
                         mpvideo\_mode.c  
                         mpvideo\_msg.h  
                         mpvideo\_response.c  
                         poll\_ready.c

### A.65 /cig/include/mx\_defines.h

The mx\_defines.h file defines the following:

- Constants used for Ballistics message queue processing (MX\_DEVICE\_CLOSED, MX\_DEVICE\_TABLE\_FULL, etc.).
- The MX\_DEVICE and MESSAGE\_HEADER structures.
- The BCOPY macro (described in Appendix B).

Included By:        bal\_routines.c  
                     ballistics.h  
                     config\_ballistics.c  
                     download\_bvols.c  
                     load\_modules.c  
                     mx\_\*.c (all files)  
                     open\_dbase.c  
                     rowcol\_rd.c  
                     simulation.c

#### **A.66 /cig/include/mx2\_defines.h**

The mx2\_defines.h file defines constants used for MPV message queue processing. It includes constants for the MX buffer status (MX\_DEVICE\_CLOSED, MX\_DEVICE\_TABLE\_FULL, etc.) and message status (MX\_MESSAGE\_SKIPPED, MX\_MESSAGE\_PUSHED, etc.).

This file is also located in the /cig/other/src/force directory.

Included By:        bootmpv.c  
                     f0\_\*.c (all files)  
                     f1\_process\_messages.c  
                     gos\_mpvio.c  
                     loadmpv.c  
                     mpvideo\_laser.c  
                     mpvideo\_lut.c  
                     mpvideo\_man.c  
                     mpvideo\_mode.c  
                     mpvideo\_response.c  
                     mx2\_hword.c  
                     mx3\_hword.c

#### **A.67 /cig/include/overlay3d\_struct.h**

The overlay3d\_struct.h file provides the typedefs for the M1 gun overlay, M2 gun overlay, gun barrel overlay, calibration overlay, and overlay parameters structures.

This file includes the standard.h and poly\_struct.h files.

Included By:        dtp\_trav2.c  
                     real\_timevpt.h

#### **A.68 /cig/include/ovrly\_defs.h**

The ovrly\_defs.h file defines constants used to create calibration overlays (for example, the dimensions of the frame triangles).

Included By:        flea.c  
                     gossip.c  
                     real\_time.h  
                     simulation.c

#### **A.69 /cig/include/poly\_struct.h**

The poly\_struct.h file provides the typedefs for the polygon command lines, poly info word, polygon list, and stamp list.

This file includes the standard.h and traverse\_cmd\_defs.h files.

Included By:        model\_struct.h  
                     overlay3d\_struct.h  
                     real\_timevpt.h  
                     vpi\_viewport.h

#### **A.70 /cig/include/ppm.h**

The ppm.h file contains definitions used when processing PPM messages. It defines the board values and message codes, and provides the typedef for the PPM\_DATA\_TYPE structure.

Included By:        msg\_ppm.c

#### **A.71 /cig/include/print\_msg\_extrns.h**

The print\_msg\_extrns.h file is not currently used.

Included By:        none

#### **A.72 /cig/include/rcinclude.h**

The rcinclude.h file is used by the DTP command generator and the Runtime Command Library. It does the following:

- Declares all RCL functions (rcl\_push, rcl\_pop, etc.).
- Declares address and pointer variables used by the RCL commands.
- Defines the RCL\_UNION structure.
- Defines the macros used by dtp\_trav1 and dtp\_trav2 to generate RCL commands. These macros, which are defined in Appendix B, are used to pass the appropriate data to rcl\_command, rcl\_lblcmd, rcl\_data, and rcl\_component.

Included By:        dtp\_compiler.c  
                     dtp\_funcs.c  
                     dtp\_trav1.c  
                     dtp\_trav2.c  
                     gos\_model.c  
                     rcfuncs.c

replace\_mod.c  
test\_commands.c

### A.73 /cig/include/real\_time.h

The real\_time.h file includes many of the include files used in the real-time software. The files it includes are the following:

- stdio.h
- dgi\_std.c.h
- dgi\_stdg.h
- functions.h
- sim\_cig\_if.h
- rtdb\_struct.h
- structures.h
- definitions.h
- ovrly\_defs.h
- def\_alloc.h
- mbx.h
- esifa.h
- clouds.h
- ecompiler1.h

Included By:

aam\_manager.c  
autopilot.c  
b0\_\*.c (all files)  
bal\_get\_db\_pos.c  
bal\_get\_lm\_grid.c  
bal\_routines.c  
ball\_effect\_add.c  
bx\_\*.c (all files)  
cal.c  
cig\_getm\_2d.c  
cig\_link\_2d.c  
cigsimio\_obj.c  
close\_db.c  
clouds.c  
config\_ballistics.c  
config\_color\_table.c  
config\_database.c  
db\_mcc\_setup.c  
debug\_initdr.c  
ded\_model\_trace.c  
ded\_object.c  
dl\_man.c  
dl\_man.c  
download\_bvols.c  
dtp\_emu.c  
dynamic\_demo.c  
effect\_downcount.c  
file\_control.c  
find\_field.c  
find\_gtfnc

flea\_\*.c (all files except flea\_simulate\_vehicles.c)  
fxbvtofl.c  
generic\_lm.c  
get\_sio\_data.c  
get\_tx\_lut\_index.c  
gos\_\*.c (all files except gos\_locate.c)  
host\_enet\_if.c  
host\_flea\_if.c  
host\_if\_debug.c  
host\_scsi\_if.c  
hw\_test.c  
load\_dbase.c  
load\_modules.c  
loc\_ter.c  
loc\_ter\_msg.c  
make\_bbn.c  
make\_bbn\_logo.c  
mkcal.c  
model\_demo.c  
model\_mtx.c  
msg\_calibration\_image.c  
mx\_\*.c (all files)  
open\_dbase.c  
open\_ded.c  
rowcol\_rd.c  
shot\_report.c  
show\_effect\_msg.c  
sio.c  
sload.c  
tick.c  
tick\_ppm.c  
tick\_script.c  
update\_2d.c  
update\_agpt\_2d.c  
upstart.c  
vt100.c

#### A.74 /cig/include/real\_timevpt.h

The real\_timevpt.h file includes the following files:

- stdio.h
- standard.h
- functions.h
- rtdb\_struct.h
- poly\_struct.h
- model\_struct.h
- db\_struct.h
- bal\_struct.h
- mpv\_struct.h
- vpi\_struct.h
- overlay3d\_struct.h
- sim\_cig\_if.h



- definitions.h
- mbx.h
- esifa.h
- clouds.h
- ecompiler1.h

Included By:       be\_stubs.c  
                    cig\_config.c  
                    gun\_overlays.c  
                    overlay\_setup.c

#### **A.75 /cig/include/rt\_definitions.h**

The rt\_definitions.h file defines constants used throughout the system (e.g., PI, TRUE, FALSE, NULL, and various constants useful as tolerance and threshold levels).

Included By:       bal\_real\_time.h

#### **A.76 /cig/include/rt\_macros.h**

The rt\_macros.h file defines the following macros: DEGREE\_TO\_RADIAN, TORAD, RADIANT\_TO\_DEGREE, ABSVAL, SQUARE, MAGSQ2D, MAGSQ3D, CUBE, SIGN, MAX, MIN, INIT\_MTX, FIND\_LM, FXTO881, FLTOFX, and FXTOFL. These are described in Appendix B.

Included By:       bal\_real\_time.h

#### **A.77 /cig/include/rt\_types.h**

The rt\_types.h file defines the data types used throughout the system (INT\_2, INT\_4, etc.).

Included By:       bal\_real\_time

#### **A.78 /cig/othersrc/force/rtcdefines.h**

The rtcdefines.h file defines constants used by the forcetask to read and reset the clock.

Included By:       forcetask.c

#### **A.79 /cig/include/rtdb\_struct.h**

The rtdb\_struct.h file defines the following real-time database structures:

- Database version and tag.
- Database header data.
- Database header overflow and landmark data.
- Generic module directory entry data and name.
- Model and catalog tables.

- Database directory entry.
- Load module header.
- Grid locator information.
- Fixed bvol entry
- Load module statistics.
- Floating bvol entry.

This file also defines the maximum number of models that can be put in the generic module of the runtime database, the maximum number of stamps in one unique static object definition, and the number of z values in a grid component.

Included By:

- flea.c
- flea\_bal\_opts.c
- gossip.c
- init\_sim.c
- msg\_cig\_ctl.c
- msg\_drll.c
- msg\_effect.c
- msg\_end.c
- msg\_pass\_on.c
- msg\_subsys\_mode.c
- msg\_veh\_state.c
- msg\_vflags.c"
- msg\_vport.c
- otherveh\_state.c
- real\_time.h
- real\_timevpt.h
- simulation.c
- staticveh\_remove.c
- staticveh\_state.c

#### A.80 /cig/include/sim\_cig\_defines.h

The sim\_cig\_defines.h file defines the message code for every CIG->SIM and SIM->CIG message. It also defines M\_LAST\_VALID\_MSG as the last (highest) valid message code.

Included By:

- flea\_simulate\_vehicles.c
- global\_init.c
- host\_drll\_if.c
- host\_mpv\_if.c
- host\_socket\_if.c
- init\_sim.c
- mpvideo\_response.c
- msg\_effect.c
- msg\_laser.c
- msg\_laser\_return.c
- msg\_lt\_state.c
- msg\_pass\_back.c
- msg\_ppm.c
- msg\_process\_round48.c
- msg\_syserr.c
- msg\_veh\_state.c
- msg\_vflags.c

print\_msg.c  
rtt\_init.c  
sim\_cig\_if.h

### A.81 /cig/include/sim\_cig\_if.h

The sim\_cig\_if.h file includes the files that define common structures, vehicles and effects, and all messages passed between the CIG and the Simulation Host. These files are:

if_hdr_str.h	commonly used structure definitions
if_veh_eff.h	vehicle and effect definitions
if_ctl_err.h	CIG control, error, and packet definition messages
if_init.h	initialization messages
if_tst_ctl.h	test and control messages
if_sim2cig.h	runtime SIM-to-CIG messages
if_cig2sim.h	runtime CIG-to-SIM messages

This file also includes sim\_cig\_defines.h, which defines all valid message codes.

Included By:	bal_struct.h
	cf_translator.c
	encode_routines.c
	model_struct.h
	real_time.h
	real_timevpt.h

### A.82 /cig/include/slave133\_functions.h

The slave133\_functions.h file declares the slave133\_malloc function. This file is included by ballistics.h when building the version of Ballistics that runs on a Slave board.

Included By:	ballistics.h
--------------	--------------

### A.83 /cig/include/standard.h

The standard.h file defines the standard data types (INT\_2, INT\_4, BYTE, etc.); 2-, 3-, and 4-dimensional vertex points; 4x3 and 4x4 matrices; 2- and 3-dimensional bounding boxes; red, green, blue; red, green, blue, opaque; hue, saturation, lightness; hue, saturation, lightness, opaque.

Included By:	bal_struct.h
	cnode_*.c (all files)
	db_struct.h
	dtp_*.c (all files)
	encode_routines.c
	fill_tree.c
	flagoff.c
	globs.c
	init_free.c
	linkvpt.c
	model_struct.h

- mpv\_struct.h
- mtx\_\*.c (all files)
- overlay3d\_struct.h
- path.c
- path\_init.c
- path\_query.c
- poly\_struct.h
- process\_vflags.c
- process\_vpops.c
- rcfuncs.c
- real\_timevpt.h
- trav\_tree.c
- tst\_\*.c (all files)
- u\_brmask.c
- u\_path.c
- u\_rotations.c
- u\_viewport.c
- u\_xfrm.c
- update\_mtx.c
- update\_rot.c
- vpi\_query.h
- vpi\_struct.h
- vpi\_viewport.h
- vpt\_\*.c (all files)

#### A.84 /gt/include/stdio.h

The stdio.h file is the Ready Systems standard input/output header file. It defines flags used for file i/o, defines various constants, and declares file i/o functions.

Included By:

- aa\_init.c
- backend\_\*.c (all files)
- bootforce.c
- bootmpv.c
- buffer\_errors.c
- bx147\_main.c
- bx\_task.c
- bx\_task.c
- cf\_translator.c
- cig\_2d\_setup.c
- cnode\_\*.c (all files)
- config\_database.c
- db\_mcc\_setup.c
- directory.c
- download\_bvols.c
- dtp\_\*.c (all files)
- esifa\_\*.c (all files)
- flagoff.c
- flea.c
- flea\_bal\_opts.c
- flea\_simulate\_vehicles.c
- globs.c
- gossip.c

host\_dr11\_if.c  
host\_ex\_if.c  
host\_mpv\_if.c  
host\_socket\_if.c  
init\_free.c  
init\_sim.c  
lex.c  
linkvpt.c  
load\_esifa.c  
loadmpv.c  
mkmtx\_nt.c  
mpvideo\_\*.c (all files except mpvideo\_print.c)  
msg\_\*.c (all files except msg\_calibration\_image.c)  
mtx\_concat.c  
mtx\_dump.c  
otherveh\_state.c  
path.c  
path\_init.c  
path\_query.c  
ppm\_obj.c  
pretend\_veh.c  
print\_msg.c  
real\_time.h  
real\_timevpt.h  
rtt.c  
rtt\_init.c  
shot\_report.c  
simulation.c  
staticveh\_remove.c  
staticveh\_state.c  
subsys\_cfg\_parse.c  
sys\_control.c  
tst\_\*.c (all files)  
u\_\*.c (all files)  
update\_mtx.c  
update\_rot.c  
vpt\_\*.c (all files)

#### A.85 /gt/include/strings.h

The strings.h file is a standard header file that declares all string-related functions.

Included By:	flea.c
	flea_atp.c
	flea_bal_opts.c
	flea_switches.c
	host_if_debug.c
	mx_error.c
	print_msg.c
	rtt.c
	rtt_init.c
	tick.c
	tick_ppm.c

tick\_script.c

#### A.86 /cig/include/struct\_2d.h

The struct\_2d.h file defines the window structures used by the 2-D compiler.

Included By:        global\_2d.h  
                     globfir\_2d.h

#### A.87 /cig/include/structures.h

The structures.h file defines various data structures used to process overlays and static and dynamic models. It includes typedefs for the following structures:

- Component data type (3-D point, 2-D point, and vector).
- Texture map index.
- Polygon information word.
- Polygon and stamp lists.
- Gunner, bun barrel, and calibration overlays.
- Field-of-view test table.
- Load module call tables.
- Static and dynamic tanks.
- Static and dynamic single-transform models.
- Remove static model.
- Show effects (stamp structure).
- Ballistics chord data.
- Trajectory positions and data.
- Load module-specific data.
- Grid component definition.

This file also defines the following:

- DTP data transformation commands.
- DTP data component commands.
- DTP data traversal commands.
- Ballistics and local terrain data pointers.
- Bounding plane definitions.
- Channel definitions.

Included By:        flea.c  
                     flea\_bal\_opts.c  
                     gossip.c  
                     init\_sim.c  
                     msg\_cig\_ctl.c  
                     msg\_dr11.c  
                     msg\_effect.c  
                     msg\_end.c  
                     msg\_pass\_on.c  
                     msg\_subsys\_mode.c  
                     msg\_veh\_state.c  
                     msg\_vport.c  
                     otherveh\_state.c

real\_time.h  
simulation.c  
staticveh\_remove.c  
staticveh\_state.c

#### **A.88 /cig/include/subsys\_cfg\_parse.h**

The `subsys_cfg_parse.h` file defines the maximum number of subsystems and the maximum length of a `subsys.cfg` file.

Included By:            `subsys_cfg_parse.c`

#### **A.89 /gt/include/sysdefs.h**

The `sysdefs.h` file provides system definitions for operating system versions RTOS.101 and RTOS.102. It includes the following:

- System-wide memory, resource, and software and hardware fault definitions.
- Task definitions.
- I/O control system definitions.
- VRTX return codes.
- Disk manager fault codes.
- File control system error codes.
- Special character definitions.
- 68901 equates.
- System interrupt equates.
- Definitions and structures used by `file_control`.

Included By:            `bx147_main.c`  
                         `rtt.c`  
                         `rtt_init.c`

#### **A.90 /cig/include/sysdefs2.h**

The `sysdefs2.h` file provides system definitions for operating system version FOS.100, which allows the use of high-speed disks. It includes the following:

- System-wide memory, resource, and software and hardware fault definitions.
- Task definitions.
- I/O control system definitions.
- VRTX return codes.
- Disk manager fault codes.
- File control system error codes.
- Special character definitions.
- 68901 equates.
- System interrupt equates.
- Definitions and structures used by `file_control`.

This file includes the `dgi_std.h` file.

Included By:            `getch.c`

**A.91 /cig/include/tflat.h**

The tflat.h file defines Ballistics round trajectories for a completely flat trajectory. This is a default table loaded for testing purposes.

Included By:           bx\_init.c

**A.92 /cig/include/tflat\_7k.h**

The tflat\_7k.h file defines Ballistics round trajectories for a completely flat trajectory, for use with 7000-meter viewing ranges. This is a default table loaded for testing purposes.

Included By:           bx\_init.c

**A.93 /cig/include/tflat\_fast.h**

The tflat\_fast.h file defines Ballistics round trajectories for a completely flat trajectory with a very fast fly-out. This is a default table loaded for testing purposes.

Included By:           bx\_init.c

**A.94 /cig/include/tflat\_slow.h**

The tflat\_slow.h file defines Ballistics round trajectories for a completely flat trajectory with a very slow fly-out. This is a default table loaded for testing purposes.

Included By:           bx\_init.c

**A.95 /cig/include/traverse\_cmd\_defs.h**

The traverse\_cmd\_defs.h file defines the following:

- The typedef for the texture map indices structure.
- The command\_id values for all transform data, data components, traverse data DTP commands.
- Constants used for Ballistics and local terrain traversal.
- Maximum buffer sizes.
- Bounding plane definitions.
- Channel definitions.

Included By:           poly\_struct.h

**A.96 /cig/include/u105mmsabot30hz.h**

The u105mmsabot30hz.h file defines Ballistics round trajectories for a u105mmsabot round with a 30 Hz sample rate. This is a default table loaded for testing purposes.



Included By:        bx\_init.c

#### A.97 /cig/include/u25mmheat.h

The u25mmheat.h file defines Ballistics round trajectories for a u25mmheat round with a 15 Hz sample rate. This is a default table loaded for testing purposes.

Included By:        bx\_init.c

#### A.98 /cig/libsrc/libvpt/vpi\_msgs.h

The vpi\_msgs.h file provides the typedefs for the messages that may appear in the ASCII viewport configuration file processed by tst\_ereadconfig. It also defines the message codes and provides the typedefs for the fov, resolution, and terrain feedback point structures.

Included By:        tst\_ereadconfig.c

#### A.99 /cig/include/vpi\_query.h

The vpi\_query.h file defines the typedefs for the vpt\_cnode\_info, vpt\_path\_info, and vpt\_vpt\_info structures used when querying information in the Viewport Configuration tree.

Included By:        cnode\_query.c  
                     path\_init.c  
                     path\_query.c  
                     tst\_equery.c  
                     tst\_treetrace.c  
                     vpt\_query.c

#### A.100        /cig/include/vpi\_struct.h

The vpi\_struct.h file defines the following for the Viewport Configuration CSC:

- The maximum number of configuration nodes and viewport nodes.
- The typedefs for the RTS3x3\_MTX, RTS4x3\_MTX, and ROT2x1 matrix structures.
- The typedef for the MTXUNION structure.

This file also includes standard.h and def\_mtx\_type.h.

Included By:        be\_stubs.c  
                     cnode\_\*.c (all files)  
                     dtp\_\*.c (all files)  
                     fill\_tree.c  
                     flagoff.c  
                     globs.c  
                     gos\_locate.c  
                     init\_free.c  
                     linkvpt.c

mkmtx\_nt.c  
mtx\_\*.c (all files)  
path.c  
path\_init.c  
path\_query.c  
process\_vppos.c  
real\_timevpt.h  
trav\_tree.c  
tst\_\*.c (all files except tst\_edebug.c)  
u\_brmask.c  
u\_path.c  
u\_rotations.c  
u\_viewport.c  
u\_xfrm.c  
update\_mtx.c  
update\_rot.c  
vpi\_viewport.h  
vpt\_\*.c (all files)

#### A.101      /cig/include/vpi\_viewport.h

The vpi\_viewport.h file defines the following for the Viewport Configuration CSC:

- The SUCCEED and FAIL return values.
- The TORAD and toradians macros (described in Appendix B).
- The maximum number of branch values.
- The typedefs for the system view flags and the view positions (vppos) array structures.
- The typedef for the configuration node, viewport parameters, and graphics path parameters structure.
- The typedefs for the fov vector, screen, and screen constants structures.

This file also includes standard.h, vpi\_struct.h, and poly\_struct.h.

Included By:

be\_stubs.c  
cig\_config.c  
cnode\_\*.c (all files)  
dtp\_\*.c (all files)  
fill\_tree.c  
flagoff.c  
globs.c  
gos\_locate.c  
init\_free.c  
linkvpt.c  
mtx\_concat.c  
path.c  
path\_init.c  
path\_query.c  
process\_vflags.c  
process\_vppos.c  
trav\_tree.c  
tst\_\*.c (all files except tst\_edebug.c)  
u\_brmask.c

u\_path.c  
u\_rotations.c  
u\_viewport.c  
u\_xfrm.c  
update\_mtx.c  
update\_rot.c  
vpt\_\*.c (all files)

## APPENDIX B. SYSTEM MACROS

Macros are used throughout the system to perform specialized functions. Most of the macros that are used by more than one function are defined in one of the following files:

**bx\_macros.h**

Macros used exclusively by Ballistics.

**functions.h**

Macros used throughout the real-time software.

**rfuncs.c and rcinclude.h**

Macros used by the Runtime Command Library (RCL) and the DTP Command Generator.

**rt\_macros**

Macros used throughout the real-time software.

Macros that are used in only one function are usually defined in the source file that contains the function.

Although some macros are used exclusively in one area of the system, others are used by multiple CSCs. For easy reference, all macros are described in this appendix, in alphabetical order. Each macro's description specifies where the macro is defined, which functions use it, what routines it calls, and what parameters it takes.

In section 2 of this document, macros are treated as functions — i.e., if a function uses a macro, the macro is listed in the function's Routines Called list.

### B.1 AAM1\_TO\_AAM2

The AAM1\_TO\_AAM2 macro determines (using AAM2\_OFFSET) the location in active area memory in backend 1 (AAM2) that corresponds to a given location in active area memory in backend 0 (AAM1).

The usage is AAM1\_TO\_AAM2(addr), where *addr* is the address in backend 0's active area memory.

Defined In:	kludge.h
Called By:	_handle_point_lights _update_second_active_area_memory
Routines Called:	none
Parameters:	INT_4                                  addr

## B.2 AAM2\_ADDR

The AAM2\_ADDR macro <<TBD>>.

The usage is AAM2\_ADDR(addr), where *addr* is <<TBD>>

Defined In:	functions.h process_vflags.c
Called By:	backend_set_color backend_set_thermal backend_update_view_paths cloud_mgmt cloud_update init_simulation process_vflags
Routines Called:	none
Parameters:	INT_4 addr

## B.3 AAREAD

The AAREAD macro, defined in definitions.h, is not currently used.

## B.4 ABSVAL

The ABSVAL macro, defined in definitions.h and rt\_macros.h, is not currently used.

## B.5 BCOPY

The BCOPY macro is used to copy a specified number of bytes. The usage is BCOPY(source, dest, byte\_count), where:

*source* is a pointer to the source location  
*dest* is a pointer to the destination location  
*byte\_count* is the number of bytes to be copied

BCOPY passes these parameters to bcopy, which performs the copy. The BCOPY macro has been retained for backward compatibility.

Defined In:	mx_defines.h
-------------	--------------

Called By:           b0\_add\_static\_vehicle  
                      b0\_bal\_config  
                      b0\_bvol\_entry  
                      b0\_database\_info  
                      b0\_model\_entry  
                      bx\_chord\_intersect  
                      download\_bvols  
                      mx\_push

Routines Called:     bcopy

Parameters:          WORD                   \*source  
                      WORD                   \*dest  
                      HWORD                  byte\_count

## B.6 CHECK\_FORCE

The CHECK\_FORCE macro checks to see if the Force task is running by reading the ready bit (FRONT\_RDY\_MASK) in the front-end control register (FE\_CONTROL). If Force is running, the Gossip operation is denied and the user is asked to retry later.

The usage is CHECK\_FORCE.

Defined In:          gos\_120tx.c  
                      gos\_mpv.c

Called By:           gos\_mpv

Routines Called:     printf

Parameters:          none

## B.7 CHECKROT

The CHECKROT macro is used to validate rotation values for Flea vehicles. Two versions of this macro exist.

The CHECKROT macro in fleasimulate\_vehicles.c is used to validate the rotation value for a Flea vehicle, and to change it if it is outside the range 0-360 degrees. The usage is CHECKROT(x), where x is the rotation in degrees. If x is greater than or equal to 360.0, CHECKROT subtracts 360.0. If x is less than or equal to 360.0, CHECKROT adds 360.0.

Defined In:          fleasimulate\_vehicles.c

Called By:           flea\_simulate\_vehicles

Routines Called:     none

Parameters:         <<TBD>>                                 x

The CHECKROT macro in flea\_update\_pos.c is used to validate the rotation value for a Flea vehicle. The usage is **CHECKROT(rotation, code)**, where:

*rotation* is the rotation in degrees  
*code* is 0 (flea\_vprot.x), 1 (flea\_vprot.y), or 2 (flea\_vprot.z)

This macro is called after flea\_update\_pos has already added 360.0 if the rotation value is less than 0.0, or subtracted 360.0 if the value is more than 360.0. If the *rotation* is still greater than 360.0 or less than 0.0, CHECKROT outputs an error and sets *rotation* to 0.0. The error message includes the *code* to identify which value was invalid.

Defined In:           flea\_update\_pos.c

Called By:           flea\_update\_pos

Routines Called:     none

Parameters:         <<TBD>>                                 rotation  
                       <<TBD>>                                 code

## B.8 CUBE

The CUBE macro, defined in rt\_macros.h, is not currently used.

## B.9 DART\_ENQUEUE

The DART\_ENQUEUE macro, defined in functions.h, is no longer used.

## B.10 DED\_BOUNDARY

The DED\_BOUNDARY macro shifts a specified number of words from a specified location to the *shift\_addr* passed to ded\_model\_trace by open\_ded. It verifies that there is room to shift the words, does the move, then calls PUSH\_STACK to push the address onto the DED model trace stack. The macro outputs a status message when it performs the shift, or an error if there is not enough room.

The usage is **DED\_BOUNDARY(adr, wc, str)**, where:

*adr* is the command address  
*wc* is the number of words to be moved  
*str* is a string to be output to identify the operation

Defined In: ded\_model\_trace.c

Called By: ded\_model\_trace

Routines Called: printf  
PUSH\_STACK

Parameters:	INT_4	adr
	INT_4	wc
	char	str[]

### B.11 DEGREE\_TO\_RADIAN

The DEGREE\_TO\_RADIAN macro, defined in rt\_macros.h, is not current used.

### B.12 DELETE\_ROUND

The DELETE\_ROUND macro is used by Ballistics to remove a round from the active list and put it on the free list.

The usage is **DELETE\_ROUND(dead\_round\_P)**, where *dead\_round\_P* is a pointer to the round to be deleted.

Defined In: bx\_macros.h

Called By: b0\_cancel\_round  
b0\_new\_frame  
b0\_process\_round  
b0\_round\_fired

Routines Called: none

Parameters:	ROUND_DATA	*dead_round_P
-------------	------------	---------------

### B.13 DELETE\_STAT\_VEH

The DELETE\_STAT\_VEH macro is used by Ballistics to remove a static vehicle from a load module list and put it on the free list.



The usage is **DELETE\_STAT\_VEH**(*dead\_sv\_P*, *table\_P*), where:

*dead\_sv\_P* is a pointer to the static vehicle to be deleted  
*table\_P* is a pointer to the vehicle table

Defined In:           bx\_macros.h

Called By:            b0\_delete\_static\_vehicle

Routines Called:     none

Parameters:           STAT\_VEH                               \*dead\_sv\_P  
                      STRUCT\_P\_SV                            \*table\_P

#### B.14 DOWNLOAD\_DATA

The **DOWNLOAD\_DATA** macro downloads 2-D overlay data into GSP memory. This macro is used by linkup to download the 2-D overlays after they are generated by the 2-D overlay compiler.

The usage is **DOWNLOAD\_DATA**. The macro does the following:

- Calls the **WAIT\_MPVIO** macro to wait for the ready bit in the Force front-end control register to be set, indicating that the MPV is ready to receive data.
- Sets the GSP addresses in the MPV mailbox.
- Copies the compiler data to the *data\_area*.
- Sets the *data\_count* in the MPV mailbox.
- Sets the Force command in *fe\_control* to **SUBSYS\_WRITE\_START**.

Defined In:           cig\_link\_2d.c

Called By:            linkup

Routines Called:     **WAIT\_MPVIO**

Parameters:           none

#### B.15 dtp.\* (DTP Macros)

Macros are used by the DTP Command Generator functions (and some testing functions in Gossip) to interface to the Runtime Command Library (RCL). The macros call RCL routines to generate the actual commands that are downloaded to the hardware.

Each DTP hardware command has one or more supporting macros. The macro called by the DTP Command Generator functions depends on the desired command, whether a label is being used, and whether relative or absolute addressing is being used.

The following table lists each DTP macro and identifies its parameters, calling routines, and called routines. It also identifies the DTP command generated by RCL for each macro. Detailed descriptions of the hardware commands are beyond the scope of this document.

Defined In: rcinclude.h

Called By: see table below

Routines Called: see table below

Parameters: see table below

Macro (parameters)	DTP Hardware Command Generated	Called By	Routines Called
dtp_bcn(label, mask, channel_data_offset)	Branch Channel Non-Zero	test_commands	rcl_iblcmd
dtp_bcnr(label, mask, channel_data_offset)	Branch Channel Non-Zero Relative	test_commands	rcl_iblcmd
dtp_bcnrs(aam_address, mask, channel_data_offset)	Branch Channel Non-Zero Relative	none	rcl_command
dtp_bcns(aam_address, mask, channel_data_offset)	Branch Channel Non-Zero	none	rcl_command
dtp_bcz(label, mask, channel_data_offset)	Branch Channel Zero	test_commands	rcl_iblcmd
dtp_bczr(label, mask, channel_data_offset)	Branch Channel Zero Relative	test_commands	rcl_iblcmd
dtp_bczrs(aam_address, mask, channel_data_offset)	Branch Channel Zero Relative	none	rcl_command
dtp_bczs(aam_address, mask, channel_data_offset)	Branch Channel Zero	none	rcl_command
dtp_bdgr(label, cos_squared)	Branch DOT Greater Than Relative	test_commands	rcl_iblcmd
dtp_bdgrs(pc_offset, cos_squared)	Branch DOT Greater Than Relative	none	rcl_command
dtp_bdlr(label, cos_squared)	Branch DOT Less Than Relative	replace_mod, test_commands	rcl_iblcmd
dtp_bdlrs(pc_offset, cos_squared)	Branch DOT Less Than Relative	none	rcl_command
dtp_bgn(label, mask)	Branch Generic Non-Zero	test_commands	rcl_iblcmd

dtb_bgns(aam_address, mask)	Branch Generic Non-Zero	none	rcl_command
dtb_bgz(label, mask)	Branch Generic Zero	test_commands	rcl_iblcmd
dtb_bgzs(aam_address, mask)	Branch Generic Zero	none	rcl_command
dtb_blm(dtp_viewpoint_address, dtp_result_address, x_multiplier, y_multiplier)	Base Load Module Calculation	dtp_trav2	rcl_command
dtb_bnz(label, mask, dtp_address)	Branch Non-Zero	dtp_trav1, dtp_trav2, test_commands	rcl_iblcmd
dtb_bnzs(label, mask, dtp_address)	Branch Non-Zero Relative	test_commands	rcl_iblcmd
dtb_bnzs(aam_address, mask, dtp_address)	Branch Non-Zero Relative	none	rcl_command
dtb_bnzrs(aam_address, mask, dtp_address)	Branch Non-Zero Relative	none	rcl_command
dtb_bpc()	Bounding Plane Normals Calculation	dtp_trav2	rcl_command
dtb_bpcx()	Bounding Plane Normals Calculation TX	none	rcl_command
dtb_bru(label)	Branch Unconditionally	dtp_trav1, dtp_trav2, replace_mod, test_commands	rcl_iblcmd
dtb_brur(label)	Branch Unconditionally Relative	test_commands	rcl_iblcmd
dtb_brurs(pc_offset)	Branch Unconditionally	none	rcl_command
dtb_brus(aam_address)	Branch Unconditionally Relative	dtp_trav1, dtp_trav2	rcl_command
dtb_brz(label, mask, dtp_address)	Branch Zero	dtp_trav2, replace_mod, test_commands	rcl_iblcmd
dtb_brzr(label, mask, dtp_address)	Branch Zero Relative	test_commands	rcl_iblcmd
dtb_brzrs(pc_offset, mask, dtp_address)	Branch Zero Relative	none	rcl_command
dtb_brzs(aam_address, mask, dtp_address)	Branch Zero	none	rcl_command
dtb_dot(vx, vy, vz)	Dot Product	replace_mod, test_commands	rcl_command
dtb_elm()	End Load Module	test_commands	rcl_command
dtb_end()	End Current Path	dtp_trav1, dtp_trav2, test_commands	rcl_command
dtb_fov(label, radius)	Field of View Test	replace_mod, test_commands	rcl_iblcmd
dtb_fovr(label, radius)	Field of View Test Relative	test_commands	rcl_iblcmd
dtb_fovrs(pc_offset, radius)	Field of View Test Relative	none	rcl_command

dtb_fovs(aam_address, radius)	Field of View Test	none	rcl_command
dtb_gdc(label, centroid_x, centroid_y, centroid_z, asid)	Generic Data Call	none	rcl_iblcmd
dtb_gdci(label, centroid_x, centroid_y, centroid_z, asid, dptr)	Generic Data Call	test_commands	rcl_iblcmd
dtb_gdcir(label, centroid_x, centroid_y, centroid_z, asid, dptr)	Generic Data Call Relative	none	rcl_iblcmd
dtb_gdcirs(aam_address, centroid_x, centroid_y, centroid_z, asid, dptr)	Generic Data Call Relative	none	rcl_command
dtb_gdcis(aam_address, centroid_x, centroid_y, centroid_z, asid, dptr)	Generic Data Call	none	rcl_command
dtb_gdcn(label, centroid_x, centroid_y, centroid_z)	Generic Data Call	none	rcl_iblcmd
dtb_gdcnr(label, centroid_x, centroid_y, centroid_z)	Generic Data Call Relative	none	rcl_iblcmd
dtb_gdcnrs(aam_address, centroid_x, centroid_y, centroid_z)	Generic Data Call Relative	none	rcl_command
dtb_gdcns(aam_address, centroid_x, centroid_y, centroid_z)	Generic Data Call	none	rcl_command
dtb_gdcr(label, centroid_x, centroid_y, centroid_z, asid)	Generic Data Call Relative	test_commands	rcl_iblcmd
dtb_gdcrs(aam_address, centroid_x, centroid_y, centroid_z, asid)	Generic Data Call Relative	none	rcl_command
dtb_gdcs(aam_address, centroid_x, centroid_y, centroid_z, asid)	Generic Data Call	none	rcl_command
dtb_gr(offset)	Generic Return	outahere, replace_mod, test_commands	rcl_command
dtb_lmi(label, radius)	Load Module In Field of View Test	test_commands	rcl_iblcmd
dtb_lmir(label, radius)	Load Module In Field of View Test Relative	test_commands	rcl_iblcmd
dtb_lmirs(pc_offset, radius)	Load Module In Field of View Test Relative	none	rcl_command
dtb_lmis(aam_address, radius)	Load Module In Field of View Test	none	rcl_command
dtb_lod(label, range_squared)	Level of Detail Test	replace_mod, test_commands	rcl_iblcmd
dtb_lodr(label, range_squared)	Level of Detail Test Relative	test_commands	rcl_iblcmd
dtb_lodrs(pc_offset, range_squared)	Level of Detail Test Relative	none	rcl_command
dtb_lods(aam_address, range_squared)	Level of Detail Test	none	rcl_command
dtb_lwd(label, dtb_address, word_count)	Load Words	dtb_trav1, test_commands	rcl_iblcmd
dtb_lwdr(label, dtb_address, word_count)	Load Words Relative	none	rcl_iblcmd

dtl_lwdrs(pc_offset, dtp_address, word_count)	Load Words Relative	none	rcl_command
dtl_lwds(aam_address, dtp_address, word_count)	Load Words	dtp_trav1, dtp_trav2	rcl_command
dtp_mml(dtp_address_a, dtp_address_b, dtp_address_c)	Matrix Multiply Local (A*B=>C)	none	rcl_command
dtp_mmpre(dtp_address_a, dtp_address_b, dtp_address_c)	Matrix Multiply Pre (A*B=>C)	none	rcl_command
dtp_mmpst(dtp_address_a, dtp_address_b, dtp_address_c)	Matrix Multiply Post (A*B=>C)	dtp_trav1, dtp_trav2	rcl_command
dtp_mwd(dtp_address_a, dtp_address_b, word_count)	Move Words	dtp_trav1	rcl_command
dtp_ngc(centroid_x, centroid_y, centroid_z)	Non-Generic Centroid	test_commands	rcl_command
dtp_oio(output_offset, word_count)	Output Indirect Offset	test_commands	rcl_command
dtp_oos(output_offset, word_count, stack_offset)	Output Offset Stack	test_commands	rcl_command
dtp_osd(label)	Output Single Word Direct	dtp_trav2	rcl_iblcmd
dtp_osds(aam_address)	Output Single Word Direct	none	rcl_command
dtp_owd(label, word_count)	Output Words Direct	dtp_trav2, outahere, replace_mod, test_commands	rcl_iblcmd
dtp_owds(aam_address, word_count)	Output Words Direct	dtp_trav2	rcl_command
dtp_owdsc(label, end_label)	Output Words Direct - Set Count	outahere, test_commands	rcl_iblcmd
dtp_owo(aam_address_offset, word_count)	Output Words Offset	replace_mod, test_commands	rcl_command
dtp_owr(label, word_count)	Output Words Relative	test_commands	rcl_iblcmd
dtp_owrs(pc_offset, word_count)	Output Words Relative	none	rcl_command
dtp_owrsc(label, end_label)	Output Words Relative - Set Count	none	rcl_iblcmd
dtp_rc()	Range Calculation	replace_mod, test_commands	rcl_command
dtp_sub(label)	Subroutine Call	replace_mod, test_commands	rcl_iblcmd
dtp_subr(label)	Subroutine Call Relative	test_commands	rcl_iblcmd
dtp_subrs(pc_offset)	Subroutine Call Relative	none	rcl_command
dtp_subs(aam_address)	Subroutine Call	dtp_trav2	rcl_command
dtp_tbc(total_time)	Time Base Calculation	replace_mod, test_commands	rcl_command
dtp_tbdr(label, start_time, end_time)	Time Base Data Relative	replace_mod, test_commands	rcl_iblcmd
dtp_tbdrrs(pc_offset, start_time, end_time)	Time Base Data Relative	none	rcl_command

ntp_tbrt(label, maximum_time)	Time Branch Relative	replace_mod, test_commands	rcl_lblcmd
ntp_tbrts(pc_offset, maximum_time)	Time Branch Relative	none	rcl_command

## B.16 DUMP\_DART\_BUFFER

The DUMP\_DART\_BUFFER macro, defined in functions.h, is no longer used.

## B.17 ERRMSG

The ERRMSG macro prints an error for the Runtime Command Library (RCL) functions. The usage is **ERRMSG(a, b)**, where:

*a* is the error message text  
*b* is the name of the calling routine

Defined In: rcfuns.c

Called By: rcl\_patch\_adrs  
rcl\_pop  
rcl\_push  
rcl\_set\_cntlbl  
rcl\_set\_label

Routines Called: printf

Parameters: char a[]  
char b[]

## B.18 EXCHANGE\_FLEA\_DATA

The EXCHANGE\_FLEA\_DATA macro is used by Flea to exchange message packets with the real-time software. It generates the MSG\_END message and the message header, posts a message to the FLEA\_INPUT\_MB mailbox, then waits for a message to be posted to the FLEA\_OUTPUT\_MB mailbox.

The usage is **EXCHANGE\_FLEA\_DATA(flea\_ims, flea\_oms)**, where:

*flea\_ims* is a pointer to the Flea-to-CIG message packet  
*flea\_oms* is a pointer to the CIG-to-Flea message packet

Defined In: functions.h

Called By: flea

flea\_init\_cig\_sw

Routines Called:   sc\_pend  
                      sc\_post

Parameters:        INT\_4                   \*flea\_ims  
                      INT\_4                   \*flea\_oms

**B.19 FIND\_LM**

The FIND\_LM macro finds the load module that a given x, y location lies in. It is assumed that the point is within active area memory.

The usage is FIND\_LM(x, y, lm, inv\_width, mask, num\_per\_side), where:

*x* is the location's x coordinate  
*y* is the location's y coordinate  
*lm* is the number of the load module  
*inv\_width* is the inverse of the width of a load module  
*mask* is the mask of the number of load module blocks per side  
*num\_per\_side* is the number of load modules per side of AAM

Defined In:        functions.h  
                      rt\_macros.h

Called By:         bal\_get\_db\_pos  
                      ball\_effect\_add  
                      bx\_probe  
                      gos\_bal\_query  
                      otherveh\_state  
                      show\_effect\_msg  
                      staticveh\_remove  
                      staticveh\_state

Routines Called:   none

Parameters:        INT\_4                   x  
                      INT\_4                   y  
                      INT\_4                   lm  
                      REAL\_4                 inv\_width  
                      INT\_4                   mask  
                      HWORD                 num\_per\_side

**B.20 FLTOFX**

The FLTOFX macro, defined in rt\_macros.h, is no longer used.

**B.21 FREE\_LM\_CACHE**

The **FREE\_LM\_CACHE** macro, when given a load module in the Ballistics database cache, puts the bounding volumes in that load module on the free bvol list, and puts the polygons in that load module on the free polygon lists.

The usage is **FREE\_LM\_CACHE(lm\_dir)**, where *lm\_dir* is a load module in the cache.

Defined In:	bx_macros.h	
Called By:	b0_lm_read bx_new_bvol bx_new_poly	
Routines Called:	none	
Parameters:	LM_CACHE_ENTRY	*lm_dir

**B.22 FXTO881**

The **FXTO881** macro converts a fixed value to floating point. The usage is **FXTO881(fxd, flt, bits)**, where:

*fxd* is the fixed point value to be converted  
*flt* is the floating point value (result)  
*bits* is the number of fractional bits in the fixed point number

Defined In:	functions.h rt_macros.h	
Called By:	bx_get_lm_data esifa_laser_return fxbvtofl fxbvtofl_020 fxbvtofl_dart local_terrain mpvideo_response	
Routines Called:	none	
Parameters:	INT_2 REAL_4 INT_4	fxd flt bits



**B.23 FXTOFL**

The FXTOFL macro, defined in `rt_macros.h`, is no longer used.

**B.24 GET\_CHORD\_END**

The GET\_CHORD\_END macro, defined in `bx_macros.h`, is not currently used.

**B.25 GET\_DB\_POS**

The GET\_DB\_POS macro is used by Ballistics to find the load module that corresponds to a given point in the database.

The usage is `GET_DB_POS(point_P, lm_width, inv_lm_width, lm_per_side)`, where:

*point\_P* is a pointer to the location in the database

*lm\_width* is the width of a load module

*inv\_lm\_width* is inverse of the width of a load module

*lm\_per\_side* is the number of load modules in a row or column of AAM

Defined In: `bx_macros.h`

Called By: `b0_process_chord`  
`b0_tf_vehicle_pos`  
`b0_traj_chord`  
`bx_trajectory`

Routines Called: none

Parameters:	<code>POINT_DATA</code>	<code>*point_P</code>
	<code>HWORD</code>	<code>lm_width</code>
	<code>REAL_4</code>	<code>inv_lm_width</code>
	<code>HWORD</code>	<code>lm_per_side</code>

**B.26 GET\_LB\_FROM\_LM**

The GET\_LB\_FROM\_LM macro is used by Ballistics to take a load module number and calculate the number of the load block that module is in.

The usage is `GET_LB_FROM_LM(lm, lb)`, where:

*lm* is the load module number (0 to 1023)

*lb* is the load block number (0 to 255)

Defined In:	bx_macros.h	
Called By:	b0_process_chord b0_round_fired bx_chord_intersect bx_round_tracer_position	
Routines Called:	none	
Parameters:	INT_4 INT_4	lm lb

## B.27 GLOB

The GLOB macro has no effect — GLOB(x) is defined as x. This macro was required in an earlier implementation to access global memory on a specific platform.

Defined In:	ememory_map.h memory_map.h
Called By:	_rowcol_rd autopilot bal_buffer_setup bx_task cal cig_config cloud_init cloud_mgmt cloud_placement cloud_update config_ballistics db_mcc_setup dtp_emu file_control flea_atp flea_bal_opts flea_graphics_test flea_init_cig_sw flea_switches flea_veh_control get_tx_lut_index getside gos_120tx gos_model gos_mpvio gos_system host_if_debug_tick

```

hw_test
load_modules
local_terrain
model_mtx
msg_calibration_image
msg_cig_ctl
msg_dr11_pkt_size
open_dbase
process_a_msg
sim_bal_* (all functions except sim_bal_init)
simulation
tick
update_2d
upstart

```

Routines Called: none

Parameters: none

## B.28 INCR\_COMPONENT

The **INCR\_COMPONENT** macro is used by the Runtime Command Library functions to update a component's word count, polygon count, and vertex count.

The usage is **INCR\_COMPONENT(*incr*)**, where *incr* is the count increment.

Defined In: rfuncs.c

Called By: rcl\_component  
rcl\_data

Routines Called: none

Parameters: WORD incr

## B.29 INIT\_MTX

The **INIT\_MTX** macro initializes a 4x3 matrix to the identity matrix. The last column is assumed and zeroes are assumed loaded. This routine is used to initialize the matrices for all static and dynamic vehicles on start-up.

The usage is **INIT\_MTX(*matrix*)**, where *matrix* is the model's transformation matrix.

Defined In: functions.h  
rt\_macros.h

Called By: cloud\_init  
dl\_setup

Routines Called: none

Parameters: REAL\_4 matrix

### B.30 MAGSQ2D

The MAGSQ2D macro, defined in rt\_macros.h, is not currently used.

### B.31 MAGSQ3D

The MAGSQ3D macro, defined in rt\_macros.h, is not currently used.

### B.32 MALLOC

The MALLOC macro is used to allocate memory for Ballistics structures. MALLOC calls slave133\_malloc if Ballistics is running on a slave board; otherwise it calls the standard malloc routine.

The usage is MALLOC(size), where *size* is the amount of memory to be allocated.

Defined In: bx\_defines.h

Called By: b0\_add\_traj\_table  
b0\_database\_info

Routines Called: malloc  
slave133\_malloc

Parameters: int size

### B.33 MAX

The MAX macro, defined in rt\_macros.h, is not currently used.

### B.34 MIN

The MIN macro, defined in rt\_macros.h, is not currently used.

### B.35 NEW\_ROUND

The NEW\_ROUND macro is used by Ballistics to get a new round from the free list and set a pointer to it.

The usage is NEW\_ROUND(*new\_round\_P*), where *new\_round\_P* is the pointer for the new round.

Defined In:           bx\_macros.h

Called By:            b0\_process\_round  
                      b0\_round\_fired

Routines Called:     none

Parameters:           ROUND\_DATA                               \*new\_round\_P

### B.36 NEW\_STAT\_VEH

The NEW\_STAT\_VEH macro is used by Ballistics to get a static vehicle from the free list and put it in the vehicle table list of a specified load module.

The usage is NEW\_STAT\_VEH (*veh\_table\_P*, *new\_sv\_P*), where:

*veh\_table\_P* is a pointer to the vehicle table  
*new\_sv\_P* is the pointer to the new vehicle

*new\_sv\_P* is set to NULL if no pointers are available (i.e., the maximum number of static vehicles has been reached).

Defined In:           bx\_macros.h

Called By:            b0\_add\_static\_vehicle

Routines Called:     none

Parameters:           STRUCT\_P\_SV                               \*veh\_table\_P  
                      STAT\_VEH                                    \*new\_sv\_P

### B.37 OPEN\_FLEA\_DATA

The OPEN\_FLEA\_DATA macro is used by Flea to obtain the file descriptors for the input and output channels for Flea-CIG communications. The macro waits for a message to be posted to the FLEA\_OUTPUT\_MB mailbox, then sets pointers to the input and output packet buffers.

The usage is OPEN\_FLEA\_DATA(input\_ptr, output\_ptr), where:

*input\_ptr* is a pointer to the Flea-to-CIG message packet buffer  
*output\_ptr* is a pointer to the CIG-to-Flea message packet buffer

Defined In: functions.h

Called By: flea

Routines Called: sc\_pend

Parameters: INT\_4 \*input\_ptr  
 INT\_4 \*output\_ptr

### B.38 OUTPUT\_MESSAGE

The OUTPUT\_MESSAGE macro, used by Flea, copies messages and message headers to the output message buffer. This macro is defined and used in two different places.

The OUTPUT\_MESSAGE macro defined in cf\_translator.c is used by config\_translator and the process\_\* functions to copy messages to a temporary message buffer (output\_msg\_buffer). The usage is OUTPUT\_MESSAGE(message), where *message* is the message or message header to be copied. The macro also keeps track of the size of the output message buffer.

Defined In: cf\_translator.c

Called By: config\_translator  
 process\_2d\_setup  
 process\_2d\_setup  
 process\_add\_traj\_table  
 process\_agl\_setup  
 process\_ammo\_define  
 process\_cig\_ctl  
 process\_configtree\_node  
 process\_define\_tx\_mode  
 process\_dr11\_pkt\_size  
 process\_file\_description

```

process_lt_state
process_lt_state
process_overlay_setup
process_ppm_display_mode
process_ppm_display_offset
process_ppm_pixel_location
process_ppm_pixel_state
process_sio_close
process_sio_init
process_tf_init_hdr
process_tf_init_pt
process_tf_state
process_traj_entry
process_viewport_state

```

Routines Called:      bcopy

Parameters:            <message structure>            message

The OUTPUT\_MESSAGE macro defined in fleas\_init\_cig\_sw is used by that function to copy messages from the temporary buffer (config\_msgs\_P) to the message packet (fleas\_img) that will be sent to the real-time software at the next exchange. The macro calculates the size of each message it adds and increments the packet size accordingly.

The usage is OUTPUT\_MESSAGE(message\_structure), where *message\_structure* is the message structure to be copied.

Defined In:            fleas\_init\_cig\_sw.c

Called By:            fleas\_init\_cig\_sw

Routines Called:      bcopy

Parameters:            <message structure>            message\_structure

### B.39 PAGE\_FORMAT

The PAGE\_FORMAT macro is used by the Ballistics portion of Gossip to handle displays that exceed one page (16 lines).

The usage is PAGE\_FORMAT(*lines*), where *lines* is the number of lines in the display.

Defined In:            bx\_probe.c  
                         gos\_bal\_query.c

Called By:	bx_probe gos_bal_query	
Routines Called:	printf scanf	
Parameters:	INT	lines

#### **B.40 poly.\* (Poly Processor Macros)**

Macros are used by the DTP Command Generator functions (and some testing functions in Gossip) to interface to the Runtime Command Library (RCL). These macros call RCL routines that generate the actual commands that are downloaded to the Polygon Processor.

Each Poly Processor command has one or more supporting macros. The following table lists each Poly Processor macro and identifies its parameters, calling routines, and called routines. It also identifies the Poly Processor command generated by RCL for each macro. Detailed descriptions of the hardware commands are beyond the scope of this document.

Defined In:	rcinclude.h
Called By:	see table below
Routines Called:	see table below
Parameters:	see table below



Macro (parameters)	Poly Processor Command Generated	Called By	Routines Called
poly_ab(alpha_0, beta_0, alpha_1, beta_1)	Alpha Betas	none	rcl_data
poly_bvc(ballistics_bit, local_terrain_bit)	Bounding Volume Component	none	rcl_component
poly_efs(label, number_of_frames)	Effect Stage	none	rcl_iblcmd
poly_efsr(label, number_of_frames)	Effect Stage Relative	none	rcl_iblcmd
poly_flu()	Flush	dtp_trav1, test_commands	rcl_command
poly_fsw()	Form Stamp Words	dtp_trav2	rcl_command
poly_gc(ballistics_bit, local_terrain_bit)	Grid Component	none	rcl_component
poly_inf(information_word)	Info Word	double_lite, single_lite, test_commands, triple_lite, vasi_lite	rcl_data
poly_lmf(matrix_pointer)	Load Matrix Full	none	rcl_command, rcl_stuff_data
poly_lsc(x, y, z, w)	Load Screen Constants	none	rcl_command
poly_mmf(matrix_pointer)	Matrix Multiply Full	replace_mod, test_commands	rcl_command, rcl_stuff_data
poly_pc(ballistics_bit, local_terrain_bit)	Poly Component	double_lite, single_lite, test_commands, triple_lite, vasi_lite	rcl_component
poly_poly(poly_info_word, vertex_list, alpha, beta)	Polygon Entry	none	rcl_data
poly_rm1()	Recall Matrix 1	dtp_trav2, replace_mod, test_commands	rcl_command
poly_rm2()	Recall Matrix 2	none	rcl_command
poly_rm3()	Recall Matrix 3	none	rcl_command
poly_rm4()	Recall Matrix 4	none	rcl_command
poly_sc(ballistics_bit, local_terrain_bit)	Stamp Component	none	rcl_component
poly_sci(ballistics_bit, local_terrain_bit, stamp_info_word, stamp_half_width, stamp_height)	Stamp Component Incomplete	none	rcl_component, rcl_data
poly_sec(ballistics_bit, local_terrain_bit)	Special Effect Component	none	rcl_component
poly_sm1()	Save Matrix 1	dtp_trav2	rcl_command
poly_sm2()	Save Matrix 2	none	rcl_command
poly_sm3()	Save Matrix 3	none	rcl_command
poly_sm4()	Save Matrix 4	none	rcl_command

poly_stamp(stamp_info_word, stamp_half_width,, stamp_height, stamp_center_x, stamp_center_y, stamp_center_z)	Stamp List Entry	none	rcl_data
poly_tog()	Channel Toggle	dtp_trav2	rcl_command
poly_vtxe(x_value, y_value, z_value)	Vertex List Entry	double_lite, single_lite, test_commands, triple_lite, vasi_lite	rcl_data
poly_vtxl(index_0, index_1, index_2, index_3)	Vertex List	double_lite, single_lite, test_commands, triple_lite, vasi_lite	rcl_data

#### B.41 POP\_STACK

The POP\_STACK macro pops the next address off the DED model trace stack. The address is set to 0 if the stack is empty.

The usage is POP\_STACK.

Defined In: ded\_model\_trace.c

Called By: ded\_model\_trace

Routines Called: none

Parameters: none

#### B.42 PRINTD4

The PRINTD4 macro prints a 32-bit word in hexadecimal and decimal format. The address at which to start printing is in the pointer variable *pntr2*.

The usage is PRINTD4().

Defined In: gos\_memory.c

Called By: gos\_memory

Routines Called: printf

Parameters: none

### B.43 PRINTD8

The PRINTD8 macro prints a double in hexadecimal and decimal format. The address at which to start printing is in the pointer variable *pntr2*.

The usage is **PRINTD8()**.

Defined In: gos\_memory.c

Called By: gos\_memory

Routines Called: printf

Parameters: none

### B.44 PRINTEX4

The PRINTEX4 macro prints a 32-bit word in hexadecimal format. The address at which to start printing is in the pointer variable *pntr2*.

The usage is **PRINTEX4()**.

Defined In: gos\_memory.c

Called By: gos\_memory

Routines Called: printf

Parameters: none

### B.45 PRINTEX8

The PRINTEX8 macro prints a 64-bit word in hexadecimal format. The address at which to start printing is in the pointer variable *pntr2*.

The usage is **PRINTEX8()**.

Defined In: gos\_memory.c

**Called By:** gos\_memory

Routines Called: printf

Parameters: none

## B.46 PUSH\_STACK

The **PUSH\_STACK** macro pushes an address onto the DED model trace stack, if there is room left in the stack.

The usage is **PUSH STACK(*adr*)**, where *adr* is the address to be added to the stack.

**Defined In:** `ded_model_trace.c`

Called By: DED\_BOUNDARY

Routines Called: none

Parameters: INT 4 adr

### B.47 RADIANT TO DEGREE

The RADIANT\_TO\_DEGREE macro, defined in rt\_macros.h, is not currently used.

### B.48 ROOM4LABEL

The ROOM4LABEL macro verifies that there is room in the RCL stack to add a label. If the stack does not have room for the label, the macro outputs an error.

The usage is **ROOM4LABEL**(store, loc), where:

*store* is the location to store the address  
*loc* is the label to set with the AAM location

**Defined In:** `rcfuncs.c`

Called By:        rcl\_iblcmd  
                     rcl\_set\_cntlbl  
                     rcl\_set\_label

Routines Called:     ERRMSG  
                         printf

Parameters:           WORD                           \*store  
                         WORD                           m

#### B.49 ROOMCHECK

The ROOMCHECK macro verifies that there is enough space for a new RCL command. The usage is ROOMCHECK(name, wd\_cnt), where:

*name* is a pointer to the routine name  
*wd\_cnt* is the number of command WORDs

The function outputs an error if space is insufficient.

Defined In:           rcfuncs.c

Called By:            rcl\_command  
                         rcl\_component  
                         rcl\_iblcmd

Routines Called:     ERRMSG

Parameters:           char                           \*name  
                         WORD                           wd\_cnt

#### B.50 SEND\_TF\_INFO

The SEND\_TF\_INFO macro generates MSG\_TF\_HDR (terrain feedback header) and MSG\_TF\_PT (terrain feedback point) messages, and puts them in the outgoing message packet. The usage is SEND\_TF\_INFO().

Defined In:           bal\_routines.c

Called By:            sim\_bal\_process\_msg

Routines Called:     none

Parameters:           none

### B.51 SET\_OUT\_BITS

The SET\_OUT\_BITS macro, defined in definitions.h, is not currently used.

### B.52 SET\_OUT\_M2BITS

The SET\_OUT\_M2BITS macro, defined in definitions.h, is not currently used.

### B.53 SET\_PPM\_DISPLAY\_OFFSET

The SET\_PPM\_DISPLAY\_OFFSET macro puts new PPM display offset values into the Flea output work buffer (p\_flea\_out). It also adjusts the values if out of range. After setting the new values, it sets ppm\_display\_offset\_flag in p\_flea\_out to TRUE.

The usage is SET\_PPM\_DISPLAY\_OFFSET().

Defined In: tick\_ppm.c

Called By: tick\_ppm

Routines Called: none

Parameters: none

### B.54 SET\_PPM\_PIXEL\_LOCATION

The SET\_PPM\_PIXEL\_LOCATION macro puts new PPM pixel location values into the Flea output work buffer (p\_flea\_out). It also adjusts the values if out of range. After setting the new values, it sets ppm\_pixel\_location\_flag in p\_flea\_out to TRUE.

The usage is SET\_PPM\_PIXEL\_LOCATION().

Defined In: tick\_ppm.c

Called By: tick\_ppm

Routines Called: none

Parameters: none

**B.55 SIGN**

The SIGN macro, defined in `rt_macros.h`, is not currently used.

**B.56 SQUARE**

The SQUARE macro, defined in `rt_macros.h`, is not currently used.

**B.57 SYSERR**

The SYSERR macro adds a MSG\_SYS\_ERROR message to the output buffer and ends processing of input messages by pointing to a dummy end statement.

The usage is **SYSERR(error, state)**, where:

*error* is the error message

*state* is the current state of the CIG

Defined In:            `functions.h`

Called By:            `db_mcc_setup`  
                       `exchange_flea_data`  
                       `exchange_scsi_data`  
                       `exchange_scsi_data_sim`  
                       `file_control.`  
                       `get_msg_2d`  
                       `hw_test`  
                       `open_dbase`  
                       `upstart`

Routines Called:      none

Parameters:	<code>INT_2</code>	<code>error</code>
	<code>INT_2</code>	<code>state</code>

**B.58 TODEG**

The TODEG macro converts an angle from radians to degrees by multiplying the given value by 57.29578122.

The usage is **TODEG(angle)**, where *angle* is the angle in radians to be converted.

Defined In:            `flea_simulate_vehicles.c`

Called By: find\_pitch\_and\_roll

Routines Called: none

Parameters: REAL\_4 angle

## B.59 TORAD

The TORAD macro converts an angle from degrees to radians by multiplying the given value by 0.017453292.

The usage is **TORAD**(angle), where *angle* is the angle in degrees to be converted.

Defined In:

- autopilot.c
- dynamic\_demo.c
- encode\_routines.c
- flea\_agl\_terrain\_follow.c
- flea\_agpt\_switches.c
- flea\_bal\_opts.c
- flea\_db\_traverse.c
- flea\_decode\_data.c
- flea\_init\_cig\_sw.c
- flea\_ppm\_obj.c
- flea\_script.c
- flea\_simulate\_vehicles.c
- flea\_switches.c
- flea\_update\_pos.c
- flea\_veh\_control.c
- gos\_model.c
- rt\_macros.h
- tick\_ppm.c
- tick\_script.c
- update\_2d.c
- update\_agpt\_2d.c
- vpi\_viewport.h

Called By:

- autopilot
- dynamic\_demo
- flea\_init\_cig\_sw
- flea\_simulate\_vehicles
- flea\_update\_pos
- gos\_model
- mtx\_concat
- send\_gun\_overlay
- tst\_eupdate
- upd\_rotation\_values
- vpt\_path\_process
- vpt\_path\_update



vpt\_update

Routines Called: none

Parameters: INT angle

**B.60 toradians**

The **toradians** macro converts an angle into radians. The usage is **toradians(angle)**. The routine multiplies the given angle by 0.017453293.

Defined In: dig\_defines.h  
make\_bbn.c  
mkmtx\_nt.c  
model\_mtx.c  
vpi\_viewport.h

Called By: rotate\_x  
rotate\_y  
rotate\_z

Routines Called: none

Parameters: INT angle

**B.61 TRIGGER\_FORCE**

The **TRIGGER\_FORCE** macro, defined in **functions.h**, is no longer used.

**B.62 VME\_TO\_VMX**

The **VME\_TO\_VMX** macro converts a VME address to a VMX address. The usage is **VME\_TO\_VMX(addr)**, where *addr* is the VME address to be converted.

Defined In: cal.c  
kludge.h  
msg\_calibration\_image.c  
msg\_effect.c  
msg\_veh\_state.c

Called By: cal  
msg\_calibration\_image  
msg\_otherveh\_state

msg\_show\_effect  
msg\_staticveh\_rem  
msg\_staticveh\_state

Routines Called: none

Parameters: INT\_4 addr

### B.63 WAIT\_FORCE

The WAIT\_FORCE macro, defined in functions.h, is no longer used.

### B.64 WAIT\_MPVIO

The WAIT\_MPVIO macro waits until the ready bit (FRONT\_RDY\_MASK) in the Force front-end control register is set, indicating that the MPV is ready to receive data. The macro outputs a message every 50,000 attempts. When the ready bit is set, the calling function continues processing.

The usage is WAIT\_MPVIO(*pmbx*), where *pmbx* is a pointer to the MPV I/O interface mailbox.

Defined In: mpvideo.h

Called By: DOWNLOAD\_DATA  
mpvideo\_boot  
mpvideo\_load  
mpvideo\_sim\_init

Routines Called: printf

Parameters: MPVIO\_INTERFACE \*pmbx

### B.65 WAIT\_MPVREPLY

The WAIT\_MPVREPLY macro waits for a message to be placed in the mpvio\_from\_buf message buffer. The macro uses mx2\_peek to check the buffer. When the status returned from mx2\_peek is MX\_MESSAGE\_PREVIEWED, the calling function continues processing.

The usage is WAIT\_MPVREPLY().

Defined In: bootmpv.c

Called By: mpvideo\_boot

Routines Called: mx2\_peek

Parameters: none

### B.66 XCLOSE

The XCLOSE macro calls the standard close routine to close files. Previously, this macro was used to call fclose for a special platform. The macro has been retained for backward compatibility.

Defined In: definitions.h

Called By: close\_db  
get\_lm  
open\_dbase  
open\_ded

Routines Called: close

Parameters: none

### B.67 XLSEEK

The XLSEEK macro calls the standard lseek routine to move to the beginning of a file. Previously, this macro was used to call fseek for a special platform. The macro has been retained for backward compatibility.

Defined In: definitions.h

Called By: ded\_load\_dtp\_code  
download\_bvols  
get\_lm  
getlmdp  
getside  
load\_dbase  
open\_dbase  
open\_ded

Routines Called: lseek

Parameters: none

### B.68 XOPEN

The XOPEN macro calls the standard open routine to open files. Previously, this macro was used to call fopen for a special platform. The macro has been retained for backward compatibility.

Defined In: definitions.h

Called By: get\_lm  
open\_dbase  
open\_ded

Routines Called: open

Parameters: none

### B.69 XREAD

The XREAD macro calls the standard read routine to read files. Previously, this macro was used to call fread for a special platform. The macro has been retained for backward compatibility.

Defined In: definitions.h

Called By: ded\_load\_dtp\_code  
download\_bvols  
get\_lm  
getlmdp  
getside  
load\_dbase  
open\_dbase  
open\_ded

Routines Called: read

Parameters: none

### B.70 XWRITE

The XWRITE macro, defined in definitions h, is no longer used.

## APPENDIX C. OPERATING SYSTEM ROUTINES

This appendix provides brief descriptions of the various operating system calls and standard C library routines used by the CIG Real-Time software.

### C.1 Special OS Service Libraries

The following table describes the system-level service routines used by the CIG Real-Time software.

Routine	Description	Called By
read_watch()	Gets the cumulative number of 500 uS ticks. Returns the number as <i>watch_count</i> .	_move_load_module_stp_to_quad_buffer, _set_up_for_next_frame, cigsimio_write, exchange_enet_data_sim, local_terrain, msg_cig_ctl, msg_end, msg_laser_request_range, msg_pass_on, process_a_msg
start_watch()	Determines which type of CPU board it is executing on, sets up the timer registers appropriately, clears the stopwatch storage areas, starts the timer, and enables the interrupts. Returns <i>board</i> .	_pend_on_frame_interrupt, cig_config, db_mcc_setup, exchange_enet_data_sim, file_control, get_msg_2d, hw_test, upstart
stop_watch()	Gets the cumulative number of 500 (100)uS ticks, stops the timer, and turns the timer interrupts off. Returns <i>watch_count</i> .	_pend_on_frame_interrupt, simulation
sysrup_off()	Ignores system/frame interrupts by moving the address of a null interrupt service routine into the 68010 exception vector space.	_pend_on_frame_interrupt, dtp_emu, gos_model, gos_system, main (rtt.c), msg_cig_ctl, s_step, sys_control_init
sysrup_on(mailbox_ptr, message)	Enables system/frame interrupts by moving the address of the interrupt service routine in to the 68010 exception vector space. Wakes up a pending routine by moving the calling task's mailbox address and the message to be returned to locations known to the isr.	dtp_emu, gos_model, gos_single_step, gos_system, host_if_debug, init_simulation, s_step

## C.2 Task Management (sc\_\*) Routines

The following table describes the routines that handle intertask mailbox communication and the creation and deletion of tasks and queues. These routines are standard Ready Systems' VRTX C interface libraries.

Routine	Description	Called By
sc_accept	Clears messages from the specified mailbox.	host_if_debug, init_simulation, msg_end, simulation
sc_delay	Pauses for a specified delay.	_rowcol_rd, bal_buffer_setup, bootforce, config_ballistics, exchange_enet_data, flea_host_if, flea_io_task, mpvideo_boot, mpvideo_load, mpvideo_sim_init, slave_cig_enet_sync
sc_fcreate	Creates a flag group.	main (rtt.c)
sc_fdelete	Deletes a flag group.	main (rtt.c)
sc_finquiry	Inquires on a flag group	main (rtt.c)
sc_fpend	Pends on a flag group.	main (rtt.c)
sc_fpost	Posts to a flag group.	poll_shutdown
sc_lock	Locks a queue to prevent concurrent use.	mx2_open, mx2_peek, mx2_push, mx2_skip, mx3_open, mx3_peek, mx3_push, mx3_skip, mx_open, mx_peek, mx_push, mx_skip
sc_pend	Waits for a message to be posted to the specified mailbox.	flea_init_cig_sw, init_simulation
sc_post	Posts a message to the specified mailbox.	_database_disable, _handle_request_local_terrain, cigsimio_frame_end, config_ballistics, exchange_flea_data, flea, flea_host_if, flea_init_cig_sw, init_simulation
sc_screate	Creates a semaphore.	main (rtt.c)
sc_sdelete	Deletes a semaphore.	main (rtt.c)
sc_sinquiry	Inquires on a semaphore's status.	poll_shutdown
sc_spend	Pends on a semaphore	main (rtt.c)
sc_spost	Posts to a semaphore.	rtt_shutdown
sc_tinquiry	Inquires on a task,	flea_io_task, main (bx147_main.c), main (rtt.c), poll_shutdown
sc_tresume	Resumes a suspended task.	flea, main (rtt.c)
sc_tsuspend	Suspends a task.	flea_io_task, main (bx147_main.c)
sc_unlock	Unlocks a locked queue.	mx2_open, mx2_peek, mx2_push, mx2_skip, mx3_open, mx3_peek, mx3_push, mx3_skip, mx_open, mx_peek, mx_push, mx_skip

### C.3 IFX File Management (ifx\_\*) Routines

The following table identifies the Ready Systems' IFX routines used by the CIG Real-Time Software to manage files and devices.

Routine	Description	Called By
ifx_close	Closes a file or device.	CloseDir, cigsimio_write, esifa_load, flea_abs_playback, gos_cigsimio, sio_close, tick_script
ifx_ioctl	Sets i/o control.	esifa_download, esifa_laser_return, esifa_load, esifa_read, esifa_read_ports, esifa_send_req, esifa_setup, esifa_write, esifa_write_ports, open_enet_interface
ifx_open	Opens a file or device.	OpenDir, cigsimio_write, esifa_load, esifa_setup, flea_abs_playback, gos_cigsimio, open_enet_interface, sio_init, stdopen, tick_script
ifx_read	Reads from a file or device.	ReadDir, esifa_load, exchange_dr11_data, exchange_dr11_data_sim, get_next_packet
ifx_sposn	Rewinds a file.	flea_abs_playback
ifx_tdelete	Deletes a task.	poll_shutdown
ifx_write	Writes to a file or device.	cigsimio_write, exchange_dr11_data, exchange_dr11_data_sim, sio_tick

## C.4 GTOS Routines

The following routines, which are called by various functions in the real-time software, are part of the GT operating system.

Routine	Description	Called By
cif_connect	Connects to an interface.	open_mpv_interface
cif_init	Initializes the communications driver.	open_mpv_interface
cif_receive	Receives a buffer.	exchange_mpv_data, exchange_mpv_data_sim
cif_send	Sends a buffer.	exchange_mpv_data, exchange_mpv_data_sim
clparse	Parses the command line entered by the user.	initialize
mpv_ncatalog	Catalogs a name.	disable_restart
mpv_nfind	Finds a name.	bal_buffer_setup, check_restart
panic	Outputs an error message.	msg_laser_return, staticveh_remove
pcreate	Creates a new task.	agpt_init, config_ballistics, main (rtt.c)
perror	Prints an error to a file.	open_socket_interface
print_allocp	Prints a free list.	tst_tree
saccept	Accepts from a socket.	open_socket_interface
sbind	Binds a name to a socket.	open_socket_interface
search	Searches for an occurrence of a string within a string.	FindField, esifa_load
seerror	Outputs IFX, VRTX, and MPV errors.	agpt_init, config_ballistics, main (bx147_main.c), main (rtt.c), poll_shutdown, rtt_shutdown
sgbuffer	Gets a buffer from the socket.	exchange_socket_data, exchange_socket_data_sim, open_socket_interface
sibuffer	Initializes a socket buffer.	open_socket_interface
slisten	Listens to a socket.	open_socket_interface
spbuffer	Puts a socket buffer.	exchange_socket_data, exchange_socket_data_sim
srecv	Receives data from a socket.	exchange_socket_data, exchange_socket_data_sim
ssend	Sends data to a socket.	exchange_socket_data, exchange_socket_data_sim
ssocket	Opens a socket.	open_socket_interface



### C.5 Standard C Runtime Libraries

The following table identifies the standard C system calls, input/output routines, and runtime libraries used by the CIG Real-Time software.

Routine	Description	Called By
acos	Computes an arc cosine.	find_pitch_and_roll
atan	Computes an arc tangent.	vpt_path_update
atoi	Converts a string to int.	host_if_disable_debug_msgs, host_if_enable_debug_msgs
bcopy	Copies a specified number of bytes from one location to another.	backend_set_branch, BCOPY, cal , cig_config, cigsimio_write, config_translator, esifa_queue_data, exchange_enet_data, exchange_enet_data_sim, flea_abs_playback, flea_bal_opts, flea_init_cig_sw, msg_calibration_image, open_enet_interface, process_* (Flea cf_translator functions), sio_write, slave_cig_enet_sync, upd_add_static_veh, upd_flea_vehicles, upd_rem_static_veh, upd_send_dynamic, upd_sio_write, upd_subsys_mode, upd_view_mag, upd_viewport_up
bzero	Zeroes out a buffer.	config_translator, process_* (Flea cf_translator functions)
calloc	Allocates memory and initializes to zero.	autopilot, cig_2d_setup, dynamic_demo, find_fn, flea, sio_init, vpt_cnode_get, vpt_cnode_set_matrix, vpt_path_get, vpt_vpt_get
close	Closes a file.	close_db, compile_2d, file_control, find_fn, flea, get_lm, gos_model , mpvideo_load, open_dbase, open_ded, sload
cmd	Sends a command to sio.	getch
cos	Calculates a cosine.	autopilot, concat_mtx, dynamic_demo, flea_init_cig_sw, flea_simulate_vehicles, flea_update_pos, gos_model, rotate_x, rotate_y, rotate_z, send_gun_overlay, tst_eupdate, upd_rotation_values, vpt_path_update
creat	Creates a file.	main (u_main2d.c)
create_sz	Creates a file with a specified size.	file_control

exit	Deletes the current task.	config_ballistics, config_color_table, config_database, config_translator, exchange_enet_data, flea, flea_host_if, flea_init_cig_sw, gossip, init_simulation, initialize, main (bx147_main.c), main (rtt.c), mpvideo_load, open_dbase, open_dr11_interface, open_enet_interface, open_scsi_interface, open_socket_interface, slave_cig_enet_sync, sys_control_init, tst_ereadconfig
fclose	Closes an I/O stream.	compile_2d, config_ballistics, config_color_table, config_database, config_translator, db_mcc_setup, flea_draw_2d, initialize, load_esifa, tst_ereadconfig
fflush	Writes all currently buffered characters in an output stream.	host_if_debug_tick
fgetc	Reads next character from stream.	p_configtree_node, process_2d_setup, process_configtree_node, read_a_keyword, remove_comment_lines, remove_white_space
fgets	Reads next line from stream.	config_ballistics, config_color_table, config_database, flea, flea_draw_2d, get_msg_2d, load_esifa, process_agl_setup, process_cig_ctl, process_configtree_node, process_define_tx_mode, process_file_description, process_overlay_setup, process_tf_init_hdr, process_tf_init_pt, process_viewport_state, remove_comment_lines
fopen	Opens an I/O stream.	config_ballistics, config_color_table, config_database, config_translator, db_mcc_setup, flea, flea_draw_2d, initialize, load_esifa, main, tst_ereadconfig
fprintf	Writes using format to specified stream.	flea, flea_printf, print_msg_*
fputc	Writes character to stream.	print_msg_sio_write, slave_cig_enet_sync
fread	Reads specified number of characters from stream.	next_char
free	Frees allocated memory.	autopilot, b0_add_traj_table, backend_setup, bx_reset, cig_2d_setup, close_db, ded_uninit, download_bvols, dtp_compiler, dynamic_demo, esifa_setup, find_fn, flea, flea_init_cig_sw, mem_check, msg_calibration_image, open_dbase, open_ded, sio_close, sio_init, vpt_tree_free

fscanf	Reads from stream using format.	p_configtree_node, p_overlay_setup, p_terrain_feedback, p_viewport_state, process_* (Flea cf_translator functions), read_a_keyword, setup_p_terrain_feedback, tst_ereadconfig
getenv	Get string value associated with environment variable	pretend_veh
gets	Gets a string.	get_sio_write_data, gos_mpvio
isalpha	Tests character to see if letter.	get_thing
isdigit	Tests character to see if digit.	flea_draw_2d, get_thing
isprint	Tests character to see if printable.	flea_agpt_locations, flea_agpt_switches, flea_atp, flea_bal_opts, flea_graphics_test, flea_switches, flea_veh_control, gossip_tick, tick, tick_ppm, tick_script
isspace	Tests character to see if space, tab, carriage return, newline, vertical tab, or formfeed.	flea_agpt_locations, flea_agpt_switches, flea_atp, flea_bal_opts, flea_draw_2d, flea_graphics_test, flea_switches, flea_veh_control, gossip_tick, tick, tick_ppm, tick_script
lseek	Moves the read/write pointer.	ded_load_dtp_code, download_bvols, file_control, get_lm, getlmdp, getside, gos_model, load_dbase, open_dbase, open_ded
malloc	Allocates memory.	b0_add_traj_table, b0_database_info, backend_setup, ded_setup, download_bvols, dtp_compiler, esifa_setup, flea, flea_init_cig_sw, host_if_debug_init, mem_check, mpvideo_setup, msg_calibration_image, open_dbase, open_ded, sio_init
max	Determines the largest of a set of values.	exchange_enet_data, exchange_enet_data_sim, slave_cig_enet_sync
min	Determines the smallest of a set of values.	mpvideo_pass_on
open	Opens a file.	file_control, find_fn, flea, get_lm, gos_model, gossip, mpvideo_load, open_dbase, open_ded, open_dr11_interface, open_scsi_interface, sload
outhexl	Outputs hex data.	b0_delete_static_vehicle, b0_state_control, b0_traj_chord, b0_traj_entry, bx_task
printf	Writes to stdout.	(used extensively throughout system)
puts	Writes to stdout.	b0_bvol_entry, b0_delete_static_vehicle, b0_inapp_message, b0_print, b0_state_control, b0_traj_chord, b0_traj_entry, b0_undefined_message, bx_task

read	Reads a file.	ded_load_dtp_code, download_bvols, exchange_scsi_data, exchange_scsi_data_sim, file_control, get_char, get_lm, getlmdp, getside, gos_model, load_dbase, mpvideo_load, open_dbase, open_ded
read_tty	Reads operator input.	flea, flea_getchar, gos_getchar
rewind	Repositions stream pointer to beginning of file.	flea
rsec	Reads multiple sectors from disk.	file_control
scanf	Reads from stdin.	dtp_emu, flea, flea_bal_opts, flea_draw_2d, flea_switches, flea_veh_control, gos_120tx, gos_bal_query, gos_cigsimio, gos_db_query, gos_locate, gos_memory, gos_model, gos_mpv, gos_mpvio, gos_ppm_query, gos_system, gossip_tick, host_if_disable_debug_msgs, host_if_enable_debug_msgs, main, model_demo, tick, tick_ppm, tick_script, tst_edebug, tst_equery, tst_ereadconfig, tst_eupdate, tst_tree, tst_treetrace, vptq_brvals, vptq_cnout, vptq_dynmtx, vptq_grout, vptq_vpout, vptq_vptbrout
sin	Calculates a sine.	autopilot, concat_mtx, dynamic_demo, flea_init_cig_sw, flea_simulate_vehicles, flea_update_pos, gos_model, rotate_x, rotate_y, rotate_z, send_gun_overlay, tst_eupdate, upd_rotation_values, vpt_path_update
sprintf	Places formatted string in character array.	config_database, open_scsi_interface
sqrt	Calculates a square root.	dtp_emu, find_pitch_and_roll, gos_model
sscanf	Scans input from stdin.	config_ballistics, config_color_table, config_database, flea, flea_draw_2d, get_sio_write_data, init_enet_interface, load_esifa, mpvideo_load
strcat	Concatenates two strings.	find_fn, get_sio_write_data
strcmp	Compares two strings.	config_ballistics, config_translator, db_mcc_setup, find_fn, flea, flea_draw_2d, flea_printf, msg_shell_sort, next_token, p_configtree_node, p_overlay_setup, p_terrain_feedback, p_viewport_state, process_* (Flea cf_translator functions), setup_comp_start, tst_ereadconfig

strcpy	Copies a string.	config_database, config_translator, db_mcc_setup, file_control, find_fn, flea, flea_agpt_locations, flea_agpt_locations_main_menu, flea_agpt_switches, flea_agpt_switches_main_menu, flea_atp, flea_atp_main_menu, flea_bal_opts, flea_bal_opts_main_menu, flea_graphics_test, flea_graphics_test_main_menu, flea_init_cig_sw, flea_prompt, flea_switches, flea_switches_main_menu, flea_veh_control, flea_veh_control_main_menu, gos_db_query_menu, gos_prompt, host_if_disable_debug_msgs, host_if_enable_debug_msgs, init_subsys_parser, initialize, main, parse_subsys_file, set_3d_lut_name, set_color_config_name, set_data_2d_name, set_database_name, set_ded_name, set_esifa_name, set_final_lut_name, tick_script, tick_script_main_menu, vpt_cnode_process
strlen	Returns string length.	config_database, db_mcc_setup, file_control, find_fn, flea_init_cig_sw, get_sio_write_data, gos_cigsimio, gossip, initialize, open_dbase, open_ded, setup_define_string, setup_text
strncmp	Compares first n characters of two strings.	db_mcc_setup, find_fn, mpvideo_load
strncpy	Copies first n characters of string.	find_fn
strchr	Searches string for last occurrence of character.	esifa_load
strtol	Converts an ASCII string to a long.	esifa_load, mpvideo_load
system	Executes a shell command.	file_control
tan	Calculates a tangent.	vpt_path_process, vpt_path_update, vpt_update
toupper	Converts a character to uppercase.	find_fn
unbf_getchar	Performs an unbuffered getchar.	cal, dtp_emu, gos_120tx, gos_bal_query, gos_cigsimio, gos_db_query, gos_memory, gos_model, gos_mpv, gos_mpvio, gos_ppm_query, gos_system, gossip, host_if_debug_menu, host_if_display_enabled_msgs, s_step
ungetc	Performs an unbuffered getchar.	read_a_keyword, remove_comment_lines, remove_white_space
write	Writes a specified number of bytes.	exchange_scsi_data, exchange_scsi_data_sim, file_control, gos_model, linkup

**APPENDIX D. GLOSSARY OF TERMS AND ABBREVIATIONS**

2-D	Two-dimensional.
AAM	Active area memory. Memory that contains the currently viewable database and models. AAM contains 256 terrain load modules (16 rows by 16 columns). This provides a 3500-meter viewing range, plus a 500-meter buffer, in each direction. If load module blocking is enabled, AAM is effectively quadrupled.
AGL	Above ground level. If AGL processing is enabled (via the MSG_AGL_SETUP message), the simulated vehicle's altitude above ground level is calculated and returned to the Simulation Host every frame.
ASID	Application-specific identification data. ASIDs are used to add unique data (bumper numbers, smoke plume, dust cloud, etc.) to a model.
aspect ratio	The ratio of the sides (width:height) of the viewport.
backend	A 9U Channel Subassembly in the CIG. The backend contains special-purpose graphics boards that transform terrain, model, and special effects data to viewpoint space. One CIG can contain two backends.
bvol	Bounding volume. The volume of the bounding box that is used to completely enclose an object in the simulation environment.
centroid	The theoretical "center" of an object, around which the object is rotated. The centroid's coordinates are the averages of the corresponding coordinates of a given set and, for a given planar or three-dimensional figure (such as a triangle or sphere), correspond to the center of mass of a thin plate of uniform thickness and consistency or a body of uniform consistency having the same boundary.
channel	A connection to a viewport. One channel may have multiple graphics paths.
CIG	Computer Image Generation System. The process of generating a 3-D, perspective accurate scene via a computer.
clipping	Removing back-facing polygons or parts of polygons that lie partially outside the viewing pyramid.
conditional node	A node in the configuration tree that causes a branch into one of two traversal paths based on some runtime condition.
configuration tree	A structure that defines the relationship between each physical component of the simulation vehicle and the location of the viewports.

data message	Smallest data component of a packet buffer.
data message header	A message that describes the contents of a data message.
DED	Dynamic Elements Database.
double-buffer memory	Memory that contains the dynamic models built by the real-time software and processed by the hardware. Dual buffering allows for one buffer to be used by the hardware while the other is being updated by the software. The buffer used for each purpose switches each frame, so the hardware is always using the buffer updated by the software during the previous frame.
downloading	The process of transferring data from the Simulation Host to the CIG.
DR11-W	A Digital Equipment Corp. standard interface that enables the Simulation Host and the CIG processor to communicate at a high transmission rate.
DTP	Data Traversal Processor.
ESIFA	Enhanced Subsystem Interface Adapter. The subsystem card that provides a communications path between the real-time software and the other boards in the 9U, including the Pixel Processor Memory (PPM) and the Pixel Processor Tiler (PPT).
EVC	Ethernet VME Controller. The board in the CIG that allows frame interrupts onto the VME bus.
dynamic vehicle	A vehicle whose position and orientation is redefined in every frame sent by the Simulation Host.
false child	The configuration tree node branched to from a conditional node if the runtime conditions is false.
fov	Field of view. The volume of space which encompasses all objects that are visible from a specific viewpoint and view angle.
frame	Information displayed on a video monitor for a duration determined by the frame speed (e.g., 33.3 milliseconds at 30 Hz or 66.6 milliseconds at 15 Hz).
frame event	An interrupt signal given by the hardware.
frame rate	The rate at which a new image is created and displayed on the screen.
frame time	The amount of time each frame is displayed.

graphics path	A window on a viewport. A T backend has one graphics path per viewport. A TX backend may have two or four graphics paths, depending on viewport resolution. Graphics path parameters are the viewport parameters that are used to load the hardware.
GSP	Graphics System Processor. The TMS34010 graphics processor on the MPV board that generates and controls 2-D graphics.
graphics processor	First board in the graphics pipeline that processes 3-D data and converts it into 2-D screen space for the tiler, based on the input of graphics processor commands. Also called the Poly Processor.
heading	The direction the viewer is pointing.
hull transformation	Description of the position and orientation of the base of a vehicle.
Hz	Hertz; cycles per second.
load module	A unit of terrain in the terrain database, measuring 500 meters by 500 meters. Data is brought into active area memory in whole load modules only.
load module block	A structure containing four load modules (two rows by two columns, for a total size of 1000 meters by 1000 meters). Blocking load modules doubles the viewing range and quadruples the amount of terrain that can be loaded into active area memory.
lod	Level of detail. The selective reduction of model detail (polygon count) or texture map detail based on distance from the viewer.
lookup table	A table used to convert color-map addresses into the actual color values displayed.
lut	Lookup table.
matrix	A rectangular array of elements arranged in rows and columns.
matrix node	A node in the configuration tree that contains a transformation matrix. The matrices in each node in a traversal path are concatenated to generate the view of the world for the viewport represented by that path.
MCC	Management, Command, and Control. The computer on the simulation network that monitors and controls the entire simulation exercise.
model	Generally used to refer to models of arbitrary, three-dimensional objects such as buildings and vehicles.
model space	The coordinate system used to define and build a particular model. The vehicle's centroid is defined as location (0,0,0).



MPV	Micro Processor Video. The last board in the graphics pipeline in a TX backend.
My_Vehicle	The simulation vehicle.
object	All simulated models: vehicles, hidden obstacles, etc.
overlay	A two-dimensional view that is displayed on a viewport on top of the three-dimensional view of the terrain.
packet buffer	Several data messages grouped together that describe one frame time.
pitch	The angle at which the viewer is looking up or down.
pixel	Picture element. The smallest addressable element on a video screen.
point lights	Light sources that can be defined from a specific location in xyz world space. This differs from directional lighting which is defined from a certain direction and originates at infinity.
Poly Processor	See graphics processor.
polygon	A closed, planar figure bounded by straight lines and consisting of three or four vertices.
PPM	Pixel Processor Memory. A board in a T backend that assembles individual pixels from the PPT into a display matrix that is the final spatial representation of an image.
PPT	Pixel Processor Tiler. An integer processor board that tiles three- and four-sided polygons with face shading or texturing pixels.
real-time	The ability to respond rapidly, frequently, or both to an event or transaction. Also refers to the software that is used to run real-time operations.
roll	The angle which measures the amount of rotation along the viewing vector (tilt).
rotation	The process by which coordinates are rotated around a particular axis. Used to define the direction of the viewing window.
rotation matrix	A means of specifying orientation.
RCL	Runtime command library. A set of routines used to generate hardware commands for the DTP and the Poly Processor.
RTS	Rotation translation scale.

scaling	The process by which an object's coordinates are changed to effectively enlarge, reduce, or skew the object in a particular direction.
SIM	The Simulation Host computer. The computer that controls the simulated vehicle's behavior.
simulation	The process that involves a computerized model of specific, significant features of some physical or logical system or environment.
simulation vehicle	The vehicle represented by a simulated viewpoint. Also called simulated vehicle or My_Vehicle.
simulator	A simulation unit consisting of a Simulation Host, a CIG, one or more monitors, and the vehicle controls. Also called a Vehicle Simulator Unit.
static vehicle	A vehicle with no anticipated movement, tracked only when its status changes.
subsystem	See backend.
T&C	Timing and Control. Board that controls all CIG synchronization and timing.
terrain database	The database on the CIG that contains the polygons that describe the simulation terrain and all objects (houses, trees, etc.) in it.
translation	The process by which coordinates are "moved" from one location to another.
transformation	A combination of translations and rotations that convert the coordinates of a point in one coordinate system into coordinates in another coordinate system.
transformation matrix	A matrix used to describe the position and orientation of an object.
true child	The configuration tree node branched to from a conditional node if the runtime conditions is true.
vector	A straight line with a specific direction.
vertex	A point in space, the termination point of a line, or the intersection point of two or more lines.
viewpoint	The direction of view from the user's eye to the target or object being viewed.
viewport	A display screen connected to the CIG. Each viewport simulates the view of the world from a specific window of the simulated vehicle.

viewport parameters	The screen resolution, viewing range, near plane, field-of-view angles, level-of-detail multiplier, and aspect ratio of a viewport.
viewspace	The area that falls within the field of view of a viewport.
VME	Versa Module European. An industry-standard bus.
world space	The absolute coordinate system used to define the simulation area. A three-dimensional space fixed relative to the world. Location (0,0) is the southwest corner of the database.

**APPENDIX E. CROSS-REFERENCE TABLES**

This appendix contains the following cross-reference tables:

- E.1 CSUs (source files) mapped to CSCs.
- E.2 Data type names mapped to typedef locations.
- E.3 Function names mapped to source file locations, with section numbers.
- E.4 Macro names mapped to source file locations, with section numbers.

## E.1 CSUs Mapped To CSCs

The following list shows every CSU (.c or .asm file) in the GT Real-Time Software CSCI, and identifies the CSC to which it belongs. The CSUs are listed in alphabetical order.

### CSU

aa\_init.c  
 aam\_manager.c  
 agpt\_init.c  
 agpt\_statistics.c  
 autopilot.c  
 b0\_aam\_centroid.c  
 b0\_aam\_sw\_corner.c  
 b0\_add\_static\_vehicle.c  
 b0\_add\_traj\_table.c  
 b0\_bal\_config.c  
 b0\_bvol\_entry.c  
 b0\_cancel\_round.c  
 b0\_cig\_frame\_rate.c  
 b0\_database\_info.c  
 b0\_delete\_static\_vehicle.c  
 b0\_delete\_traj\_table.c  
 b0\_error\_detected.c  
 b0\_inapp\_message.c  
 b0\_lm\_read.c  
 b0\_model\_directory.c  
 b0\_model\_entry.c  
 b0\_new\_frame.c  
 b0\_print.c  
 b0\_process\_chord.c  
 b0\_process\_round.c  
 b0\_round\_fired.c  
 b0\_state\_control.c  
 b0\_status\_request.c  
 b0\_tf\_init\_hdr.c  
 b0\_tf\_init\_pt.c  
 b0\_tf\_state.c  
 b0\_tf\_vehicle\_pos.c  
 b0\_traj\_chord.c  
 b0\_traj\_entry.c  
 b0\_undefined\_message.c  
 backend\_branch.c

### CSC

Backend Manager  
 CIG Configuration  
 Real-Time Processing  
 User Interface Mode  
 Stand-Alone Host Emulator  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Backend Manager

**CSU**

backend\_color.c  
 backend\_laser.c  
 backend\_man.c  
 backend\_paths.c  
 backend\_thermal.c  
 backend\_video.c  
 bal\_get\_db\_pos.c  
 bal\_get\_lm\_grid.c  
 bal\_routines.c  
 ball\_effect\_add.c  
 bbnc\_type.c  
 be\_stubs.c  
 bit\_blt.c  
 blcopy.c  
 bootforce.c  
 bootmpv.c  
 buffer\_errors.c  
 bx147\_main.c  
 bx\_bvol\_int.c  
 bx\_chord\_intersect.c  
 bx\_compute\_round.c  
 bx\_find\_vehicle.c  
 bx\_functions.c  
 bx\_get\_lm\_data.c  
 bx\_get\_lm\_grid.c  
 bx\_init.c  
 bx\_model\_int.c  
 bx\_poly\_int.c  
 bx\_probe.c  
 bx\_reset.c  
 bx\_task.c  
 bx\_tf\_pack.c  
 bx\_trajectory.c  
 cal.c  
 cf\_translator.c  
 cig\_2d\_setup.c  
 cig\_comp\_2d.c  
 cig\_config.c  
 cig\_getm\_2d.c  
 cig\_link\_2d.c  
 cigsimio\_obj.c  
 close\_db.c  
 clouds.c

**CSC**

Backend Manager  
 Backend Manager  
 Backend Manager  
 Backend Manager  
 Backend Manager  
 Backend Manager  
 Real-Time Processing  
 Real-Time Processing  
 Real-Time Processing  
 Real-Time Processing  
 CIG Configuration  
 Viewport Configuration  
 2-D Overlay Compiler  
 System Utilities  
 MPV Interface  
 MPV Interface  
 User Interface Mode  
 Task Initialization  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Ballistics Processing  
 Real-Time Processing  
 Stand-Alone Host Emulator  
 2-D Overlay Compiler  
 2-D Overlay Compiler  
 CIG Configuration  
 2-D Overlay Compiler  
 2-D Overlay Compiler  
 Real-Time Processing  
 Real-Time Processing  
 Real-Time Processing

**CSU**

cnode\_child.c  
 cnode\_get.c  
 cnode\_process.c  
 cnode\_query.c  
 cnode\_set.c  
 comp.c  
 config\_ballistics.c  
 config\_color\_table.c  
 config\_database.c  
 data\_type.c  
 db\_mcc\_setup.c  
 debug\_initdr.c  
 ded\_gm\_pool.c  
 ded\_model\_trace.c  
 ded\_object.c  
 directory.c  
 dl\_man.c  
 download\_bvols.c  
 draw\_line.c  
 dtp\_compiler.c  
 dtp\_emu.c  
 dtp\_funcs.c  
 dtp\_trav1.c  
 dtp\_trav2.c  
 dynamic\_demo.c  
 effect\_downcount.c  
 encode\_routines.c  
 esifa\_fade.c  
 esifa\_laser.c  
 esifa\_load.c  
 esifa\_man.c  
 esifa\_query.c  
 esifa\_special.c  
 esifa\_thermal.c  
 esifa\_video.c  
 exception.asm  
 f0\_3dlut\_download.c  
 f0\_3dlut\_switch.c  
 f0\_alllut\_switch.c  
 f0\_debug\_disable.c  
 f0\_debug\_enable.c  
 f0\_final\_lut\_download.c  
 f0\_final\_lut\_switch.c

**CSC**

Viewport Configuration  
 Viewport Configuration  
 Viewport Configuration  
 Viewport Configuration  
 Viewport Configuration  
 2-D Overlay Compiler  
 Real-Time Processing  
 Real-Time Processing  
 Real-Time Processing  
 Force Processing  
 Real-Time Processing  
 Real-Time Processing  
 Real-Time Processing  
 Real-Time Processing  
 User Interface Mode  
 System Utilities  
 Backend Manager, Real-Time Processing  
 Real-Time Processing  
 2-D Overlay Compiler  
 DTP Command Generator  
 User Interface Mode  
 DTP Command Generator  
 DTP Command Generator  
 DTP Command Generator  
 Stand-Alone Host Emulator  
 Real-Time Processing  
 Stand-Alone Host Emulator  
 ESIFA Interface  
 ESIFA Interface  
 ESIFA Interface  
 ESIFA Interface  
 ESIFA Interface  
 ESIFA Interface  
 ESIFA Interface  
 ESIFA Interface  
 Force Processing  
 Force Processing  
 Force Processing  
 Force Processing  
 Force Processing  
 Force Processing  
 Force Processing

**CSC**

[illegible]



**CSU**

flea\_update\_pos.c  
 flea\_veh\_control.c  
 force.asm  
 forcetask.c  
 fxbvtofl.c  
 generic\_lm.c  
 get\_sio\_data.c  
 get\_thing.c  
 get\_tx\_lut\_index.c  
 get\_vehicle\_position.c  
 getch.c  
 global\_init.c  
 globs.c  
 gos\_120tx.c  
 gos\_bal\_query.c  
 gos\_db\_query.c  
 gos\_locate.c  
 gos\_memory.c  
 gos\_model.c  
 gos\_mpv.c  
 gos\_mpvio.c  
 gos\_polys.c  
 gos\_system.c  
 gossip.c  
 gsp\_io.c  
 gun\_overlays.c  
 host\_dr11\_if.c  
 host\_enet\_if.c  
 host\_flea\_if.c  
 host\_if\_debug.c  
 host\_mpv\_if.c  
 host\_scsi\_if.c  
 host\_socket\_if.c  
 hw\_test.c  
 init\_free.c  
 init\_sim.c  
 init\_stuff.c  
 lex.c  
 linkvpt.c  
 load\_dbase.c  
 load\_esifa.c  
 load\_modules.c  
 loadmpv.c

**CSC**

Stand-Alone Host Emulator  
 Stand-Alone Host Emulator  
 Force Processing  
 Force Processing  
 Real-Time Processing  
 Real-Time Processing  
 Stand-Alone Host Emulator  
 2-D Overlay Compiler  
 Real-Time Processing  
 CIG Configuration  
 CIG Configuration  
 Real-Time Processing  
 Viewport Configuration  
 User Interface Mode  
 User Interface Mode  
 User Interface Mode  
 User Interface Mode  
 User Interface Mode  
 User Interface Mode  
 User Interface Mode  
 User Interface Mode  
 User Interface Mode  
 User Interface Mode  
 Force Processing  
 Real-Time Processing  
 Host Interface Manager  
 Host Interface Manager  
 Host Interface Manager  
 Host Interface Manager  
 Host Interface Manager  
 Host Interface Manager  
 Host Interface Manager  
 Real-Time Processing  
 Viewport Configuration  
 Real-Time Processing  
 2-D Overlay Compiler  
 Token Processing  
 Viewport Configuration  
 Real-Time Processing  
 Real-Time Processing  
 Real-Time Processing  
 MPV Interface

**CSU**

loc\_ter.c  
loc\_ter\_msg.c  
make\_bbn.c  
make\_bbn\_logo.c  
mkcal.c  
mkmtx\_nt.c  
model\_demo.c  
model\_mtx.c  
mpvideo\_laser.c  
mpvideo\_lut.c  
mpvideo\_man.c  
mpvideo\_mode.c  
mpvideo\_pass\_back.c  
mpvideo\_pass\_on.c  
mpvideo\_print.c  
mpvideo\_query.c  
mpvideo\_response.c  
msg\_calibration\_image.c  
msg\_cig\_ctl.c  
msg\_drill.c  
msg\_effect.c  
msg\_end.c  
msg\_laser.c  
msg\_laser\_return.c  
msg\_lt\_state.c  
msg\_pass\_back.c  
msg\_pass\_on.c  
msg\_ppm.c  
msg\_process\_round48.c  
msg\_subsys\_mode.c  
msg\_syserr.c  
msg\_veh\_state.c  
msg\_vflags.c  
msg\_vport.c  
mtx\_concat.c  
mtx\_dump.c  
mtx\_viewspace.c  
mx2\_hword.c  
mx3\_hword.c  
mx\_error.c  
mx\_open.c  
mx\_peek.c  
mx\_push.c

**CSC**

**Real-Time Processing**

**Message Processing**

**Real-Time Processing**

**User Interface Mode**

**Task Initialization**

**Real-Time Processing**

**Stand-Alone Host Emulator**

**Real-Time Processing**

**MPV Interface**

**MPV Interface**

**MPV Interface**

**MPV Interface**

**MPV Interface**

**MPV Interface**

**MPV Interface**

**MPV Interface**

**MPV Interface**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Message Processing**

**Viewport Configuration**

**Viewport Configuration**

**Viewport Configuration**

**Force Processing, Real-Time Processing**

**User Interface Mode**

**Ballistics Processing**

**Ballistics Processing**

**Ballistics Processing**

**Ballistics Processing**

**CSU**

mx\_skip.c  
 mx\_wcopy.c  
 nmi\_type.c  
 open\_dbase.c  
 open\_ded.c  
 otherveh\_state.c  
 oval\_rect.c  
 overlay\_setup.c  
 path.c  
 path\_init.c  
 path\_query.c  
 poll\_ready.c  
 poly.c  
 ppm\_obj.c  
 pretend\_veh.c  
 print\_msg.c  
 proc\_cmd.c  
 process\_vflags.c  
 process\_vpops.c  
 rcfuns.c  
 replace\_mod.c  
 rowcol\_rd.c  
 rt\_mailbox.c  
 rtt.c  
 rtt\_init.c  
 shot\_report.c  
 show\_effect\_msg.c  
 simulation.c  
 sio.c  
 sload.c  
 staticveh\_remove.c  
 staticveh\_state.c  
 stdopen.c  
 string.c  
 subsys\_cfg\_parse.c  
 sys\_control.c  
 test\_commands.c  
 test\_gsp.c  
 text.c  
 tick.c  
 tick\_ppm.c  
 tick\_script.c  
 trav\_tree.c

**CSC**

Ballistics Processing  
 Ballistics Processing  
 Force Processing  
 Real-Time Processing  
 Real-Time Processing  
 Real-Time Processing  
 2-D Overlay Compiler  
 CIG Configuration  
 Viewport Configuration  
 Viewport Configuration  
 Viewport Configuration  
 Force Processing  
 2-D Overlay Compiler  
 Backend Manager  
 Real-Time Processing  
 Message Processing  
 2-D Overlay Compiler  
 CIG Configuration  
 CIG Configuration  
 DTP Command Generator  
 User Interface Mode  
 Real-Time Processing  
 Real-Time Processing  
 Task Initialization  
 Real-Time Processing  
 Ballistics Processing  
 Message Processing  
 Real-Time Processing  
 Serial Device Input/Output  
 System Utilities  
 Real-Time Processing  
 Real-Time Processing  
 System Utilities  
 2-D Overlay Compiler  
 Token Processing  
 Real-Time Processing  
 User Interface Mode  
 Force Processing  
 2-D Overlay Compiler  
 Stand-Alone Host Emulator  
 Stand-Alone Host Emulator  
 Stand-Alone Host Emulator  
 Viewport Configuration

**CSU**

tst\_edebug.c  
tst\_equery.c  
tst\_ereadconfig.c  
tst\_eupdate.c  
tst\_tree.c  
tst\_treetrace.c  
u\_brmask.c)  
u\_comp\_2d.c  
u\_getm\_2d.c  
u\_link\_2d.c  
u\_main2d.c  
u\_path.c  
u\_rotations.c  
u\_viewport.c  
u\_xfrm.c  
update\_2d.c  
update\_agpt\_2d.c  
update\_mtx.c  
update\_rot.c  
upstart.c  
vpt\_get.c  
vpt\_process.c  
vpt\_query.c  
vpt\_set.c  
vpt\_update.c  
vt100.c  
window.c

**CSC**

Viewport Configuration  
Viewport Configuration  
Viewport Configuration  
Viewport Configuration  
Viewport Configuration  
Viewport Configuration  
Viewport Configuration  
2-D Overlay Compiler  
2-D Overlay Compiler  
2-D Overlay Compiler  
2-D Overlay Compiler  
Viewport Configuration  
Viewport Configuration  
Viewport Configuration  
Viewport Configuration  
Stand-Alone Host Emulator  
Stand-Alone Host Emulator  
Viewport Configuration  
Viewport Configuration  
Real-Time Processing  
Viewport Configuration  
Viewport Configuration  
Viewport Configuration  
Viewport Configuration  
Viewport Configuration  
System Utilities  
2-D Overlay Compiler

## E.2 Data Type Names Mapped To Typedefs

The following list shows the special data types used throughout the real-time software, and identifies the file(s) that contain the type definition. The type names are listed in alphabetical order.

<b>Data Type</b>	<b>Typedef Location</b>
(*MAT_PTR4)[3]	/cig/include/rt_types.h
(*MAT_PTR8)[3]	/cig/include/rt_types.h
(*PFI2)()	/cig/include/dgi_stdg.h, standard.h
(*PFI4)()	/cig/include/dgi_stdg.h, standard.h
(*PFR4)()	/cig/include/dgi_stdg.h, standard.h
(*PFR8)()	/cig/include/dgi_stdg.h, standard.h
4RGBO	/cig/include/standard.h
AAM	/cig/include/backend.h
ALLOC_POLY	/cig/include/struct_2d.h
ASID_OMODEL	/cig/include/model_struct.h, structures.h
ASID_SHOW_EFF	/cig/include/model_struct.h, structures.h
AUTOP	/cig/libsrc/libflea/autopilot.c
B1BBOX2D	/cig/include/dgi_stdg.h, standard.h
B1BBOX3D	/cig/include/dgi_stdg.h, standard.h
B1HSL	/cig/include/dgi_stdg.h, standard.h
B1HSLO	/cig/include/dgi_stdg.h, standard.h
B1MTX4X3	/cig/include/dgi_stdg.h, standard.h
B1MTX4X4	/cig/include/dgi_stdg.h, standard.h
B1P2D	/cig/include/dgi_stdg.h, standard.h
B1P3D	/cig/include/dgi_stdg.h, standard.h
B1P4D	/cig/include/dgi_stdg.h, standard.h
B1RGB	/cig/include/dgi_stdg.h, standard.h
B1RGBO	/cig/include/dgi_stdg.h, standard.h
BACKEND_OBJ	/cig/include/backend.h
BOOLEAN	/cig/include/dgi_stdg.h, rt_types.h, standard.h; /cig/othersrc/force/dgi_stdg.h
BVOL_CACHE_ENTRY	/cig/include/bx_structs.h
BVOL_ENTRY	/cig/include/bx_rtdb_structs.h, rtdb_struct.h
BX_MODEL	/cig/include/bx_rtdb_structs.h
BX_SHOW_EFF	/cig/include/bx_rtdb_structs.h
BX_TANK	/cig/include/bx_rtdb_structs.h
BYTE	/cig/include/dgi_stdg.h, rt_types.h, standard.h; /cig/othersrc/force/dgi_stdg.h
CAL_OVRLY	/cig/include/overlay3d_struct.h, structures.h
CATALOG_TABLE_STRUCT	/cig/include/rtdb_struct.h

**Data Type****Typedef Location**

CHAN_CNST	/cig/include/poly_struct.h, structures.h
CHAN_SETCMD	/cig/include/poly_struct.h, structures.h
CHORD	/cig/include/bx_structs.h
CHORD_DATA	/cig/include/bal_struct.h, structures.h
CLOUD_CONTROL	/cig/include/clouds.h
CMD	/cig/include/poly_struct.h, structures.h
CMDR_OVRLY	/cig/include/overlay3d_struct.h, structures.h
COLOR_TABLE	/cig/include/mpv_struct.h, structures.h
COMMAND_LINE1	/cig/include/bx_rtdb_structs.h, poly_struct.h, structures.h
COMMAND_LINE2	/cig/include/bx_rtdb_structs.h, poly_struct.h, structures.h
CONFIGURATION_NODE	/cig/include/vpi_viewport.h
DB_DIR_ENTRY	/cig/include/bx_rtdb_structs.h, rtdb_struct.h
DB_HDR	/cig/include/bx_rtdb_structs.h
DB_HDR_DBASE_DATA	/cig/include/rtdb_struct.h
DB_HDR_LMARKS_DATA	/cig/include/rtdb_struct.h
DB_HDR_OFLOW_DATA	/cig/include/rtdb_struct.h
DB_INFO	/cig/include/db_struct.h, structures.h
DB_PTRS	/cig/include/db_struct.h, structures.h
DB_TAG_STRUCT	/cig/include/rtdb_struct.h
DB_VERSION_STRUCT	/cig/include/rtdb_struct.h
DED_POOL	/cig/libsrc/librt/ded_gm_pool.c, open_ded.c
DEMO1	/cig/include/demo_struct.h
DIGITIZER_INTERFACE	/cig/include/dig_struct.h
DTP_CMND_INF	/cig/libsrc/librt/ded_model_trace.c
DTP_MODE_SWITCH	/cig/include/mpv_struct.h, structures.h
DTP_TRACE_STRUCT	/cig/libsrc/libgossip/ded_object.c
EDGE_FLG	/cig/include/definitions.h
EO_EFFECTS	/cig/include/structures.h
EO_OVRLY	/cig/include/structures.h
ESIFA_DOWNLOAD_QUEUE_STRUCT	/cig/libsrc/libesifa/esifa_man.c
ESIFA_OBJ	/cig/include/esifa.h
ESIFA_PORTS	/cig/include/esifa.h
EVC	/cig/libsrc/librt/sys_control.c
F_SIMIN	/cig/include/demo_struct.h
F_SIMOUT	/cig/include/demo_struct.h
FADE_DESCR	/cig/libsrc/libflea/flea_agpt_switches.c
FADE_TRANSFER	/cig/include/mpv_struct.h, structures.h
FAKE_DV	/cig/include/model_struct.h, structures.h
FIX_BVOL_ENTRY	/cig/include/bx_rtdb_structs.h, rtdb_struct.h
FORCE_ENVIRONMENT	/cig/include/force_env.h; /cig/othersrc/force/force_env.h, force_rtsw_if.h
FORCE_INTERFACE	/cig/include/mbx.h; /cig/othersrc/force/force_defines.h, mbx.h

**Data Type****Typedef Location**

FOV	/cig/include/if_hdr_str.h; cig/libsrc/libvpt/vpi_msgs.h
FOV_VECTORS	/cig/include/vpi_viewport.h
FOVTT	/cig/include/poly_struct.h, structures.h
GENLM	/cig/libsrc/librt/generic_lm.c
GM_DE	/cig/include/bx_rtdb_structs.h
GM_DIR_ENTRY_DATA	/cig/include/rtdb_struct.h
GM_DIR_ENTRY_NAME	/cig/include/rtdb_struct.h
GRAPHICS_PATH_PARAMETERS	/cig/include/vpi_viewport.h
GRID_COMP_DEF	/cig/include/bx_rtdb_structs.h, poly_struct.h, structures.h
GRID_LOC	/cig/include/bx_rtdb_structs.h, rtdb_struct.h
GUN_B	/cig/include/overlay3d_struct.h, structures.h
GUN_B_LSIDE	/cig/include/overlay3d_struct.h, structures.h
GUN_B_RSIDE	/cig/include/overlay3d_struct.h, structures.h
GUNNER_OVRLY	/cig/include/overlay3d_struct.h, structures.h
HWORD	/cig/include/dgi_std.h, rt_types.h, standard.h; /cig/othersrc/force/dgi_std.h
I2BBOX2D	/cig/include/dgi_std.h, standard.h
I2BBOX3D	/cig/include/dgi_std.h, standard.h
I2HSL	/cig/include/dgi_std.h, standard.h
I2HSLO	/cig/include/dgi_std.h, standard.h
I2IJ	/cig/include/standard.h
I2MTX4X3	/cig/include/dgi_std.h, standard.h
I2MTX4X4	/cig/include/dgi_std.h, standard.h
I2P2D	/cig/include/dgi_std.h, standard.h
I2P3D	/cig/include/dgi_std.h, rt_types.h, standard.h
I2P3D	/cig/include/dgi_std.h, standard.h
I2RGB	/cig/include/dgi_std.h, standard.h
I2RGBO	/cig/include/dgi_std.h, standard.h
I4BBOX2D	/cig/include/dgi_std.h, standard.h
I4BBOX3D	/cig/include/dgi_std.h, standard.h
I4HSL	/cig/include/dgi_std.h, standard.h
I4HSLO	/cig/include/dgi_std.h, standard.h
I4IJ	/cig/include/standard.h
I4MTX4X3	/cig/include/dgi_std.h, standard.h
I4MTX4X4	/cig/include/dgi_std.h, standard.h
I4P2D	/cig/include/dgi_std.h, standard.h
I4P3D	/cig/include/dgi_std.h, rt_types.h, standard.h
I4P4D	/cig/include/dgi_std.h, standard.h
I4RGB	/cig/include/dgi_std.h, standard.h
I4RGBO	/cig/include/dgi_std.h
INT_2	/cig/include/dgi_std.h, rt_types.h, standard.h; /cig/othersrc/force/dgi_std.h

<b>Data Type</b>	<b>Typedef Location</b>
INT_4	/cig/include/dgi_std.h, rt_types.h, standard.h; /cig/othersrc/force/dgi_std.h
IOB	/cig/libsrc/libcio/sio.c
LM_CACHE_ENTRY	/cig/include/bx_structs.h
LM_CALL1	/cig/include/poly_struct.h, structures.h
LM_CALL2	/cig/include/poly_struct.h, structures.h
LM_H	/cig/include/bx_rtdb_structs.h, rtdb_struct.h
LM_SEARCH_LIST	/cig/include/bx_structs.h
LM_STATS	/cig/include/bx_rtdb_structs.h, rtdb_struct.h
LMS_DATA	/cig/include/db_struct.h, structures.h
LT_BVOL_ENTRY	/cig/include/if_cig2sim.h
LT_POLY_ENTRY	/cig/include/if_cig2sim.h
M1_GUN_OVERLAY	/cig/include/overlay3d_struct.h, structures.h
M2_GUN_OVERLAY	/cig/include/overlay3d_struct.h, structures.h
MAT_UNIT	/cig/include/poly_struct.h, structures.h
MATRIX4[3][3]	/cig/include/rt_types.h
MATRIX8[3][3]	/cig/include/rt_types.h
MDL_ADDR_STRUCT	/cig/libsrc/libgossip/ded_object.c
MESSAGE_HEADER	/cig/include/mx_defines.h; /cig/libsrc/libgossip/mx3_hword.c; /cig/libsrc/librt/mx2_hword.c; /cig/othersrc/force/mx2_hword.c
MODE_SELECT_STRUCT	/cig/othersrc/force/fl_defines.h
MODEL_TABLE_STRUCT	/cig/include/rtdb_struct.h
MPVIDEO_CONTROL	/cig/include/mpvideo.h
MPVIDEO_OBJ	/cig/include/mpvideo.h
MPVIO_INTERFACE	/cig/include/mpvideo.h
MSG_1ROTATION	/cig/include/if_sim2cig.h
MSG_2D_SETUP	/cig/include/if_init.h
MSG_3ROTATIONS	/cig/include/if_sim2cig.h
MSG_ADD_TRAJ_TABLE	/cig/include/if_sim2cig.h
MSG_AGL	/cig/include/if_cig2sim.h
MSG_AGL_SETUP	/cig/include/if_sim2cig.h
MSG_AMMO_DEFINE	/cig/include/if_init.h
MSG_B0_AAM_CENTROID	/cig/include/bx_messages.h
MSG_B0_AAM_SW_CORNER	/cig/include/bx_messages.h
MSG_B0_ADD_STATIC_VEHICLE	/cig/include/bx_messages.h
MSG_B0_ADD_TRAJ_TABLE	/cig/include/bx_messages.h
MSG_B0_BAL_CONFIG	/cig/include/bx_messages.h
MSG_B0_BVOL_ENTRY	/cig/include/bx_messages.h
MSG_B0_CANCEL_ROUND	/cig/include/bx_messages.h
MSG_B0_CIG_FRAME_RATE	/cig/include/bx_messages.h
MSG_B0_DATABASE_INFO	/cig/include/bx_messages.h
MSG_B0_DELETE_STATIC_VEHICLE	/cig/include/bx_messages.h



**Data Type****Typedef Location**

MSG_B0_DELETE_TRAJ_TABLE	/cig/include/bx_messages.h
MSG_B0_LM_READ	/cig/include/bx_messages.h
MSG_B0_MODEL_DIRECTORY	/cig/include/bx_messages.h
MSG_B0_MODEL_ENTRY	/cig/include/bx_messages.h
MSG_B0_NEW_FRAME	/cig/include/bx_messages.h
MSG_B0_PRINT	/cig/include/bx_messages.h
MSG_B0_PROCESS_CHORD	/cig/include/bx_messages.h
MSG_B0_PROCESS_ROUND	/cig/include/bx_messages.h
MSG_B0_ROUND_FIRED	/cig/include/bx_messages.h
MSG_B0_STATE_CONTROL	/cig/include/bx_messages.h
MSG_B0_TF_INIT_HDR	/cig/include/bx_messages.h
MSG_B0_TF_INIT_PT	/cig/include/bx_messages.h
MSG_B0_TF_STATE	/cig/include/bx_messages.h
MSG_B0_TF_VEHICLE_POS	/cig/include/bx_messages.h
MSG_B0_TRAJ_CHORD	/cig/include/bx_messages.h
MSG_B0_TRAJ_ENTRY	/cig/include/bx_messages.h
MSG_B1_GLOBAL_ADDR	/cig/include/bx_messages.h
MSG_B1_HIT_RETURN	/cig/include/bx_messages.h
MSG_B1_MISS	/cig/include/bx_messages.h
MSG_B1_ROUND_POSITION	/cig/include/bx_messages.h
MSG_B1_SHOT_REPORT	/cig/include/bx_messages.h
MSG_B1_STATUS_RETURN	/cig/include/bx_messages.h
MSG_B1_TF_HDR	/cig/include/bx_messages.h
MSG_B1_TF_PT	/cig/include/bx_messages.h
MSG_CALIBRATION_IMAGE	/cig/include/if_sim2cig.h
MSG_CANCEL_ROUND	/cig/include/if_sim2cig.h
MSG_CIG_CTL	/cig/include/if_ctl_err.h
MSG_CLOUD_STATE	/cig/include/if_sim2cig.h
MSG_CREATE_CONFIGNODE	/cig/include/if_init.h; /cig/libsrc/libvpt/vpi_msgs.h
MSG_DATA_TEST	/cig/include/if_sim2cig.h
MSG_DEFINE_TX_MODE	/cig/include/if_init.h
MSG_DELETE_TRAJ_TABLE	/cig/include/if_sim2cig.h
MSG_DR11_PKT_SIZE	/cig/include/if_ctl_err.h
MSG_F0_3DLUT_DOWNLOAD	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F0_3DLUT_SWITCH	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F0_ALLLUT_SWITCH	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F0_FINAL_LUT_DOWNLOAD	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F0_FINAL_LUT_SWITCH	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h

<b>Data Type</b>	<b>Typedef Location</b>
MSG_F0_MODE_SELECT	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F0_MPV_PEEK	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F0_MPV_POKE	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F0_MPV_POKE16	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F0_MPV_TASK_CONTROL	/cig/othersrc/force/mpvideo_msg.h; /cig/include/mpvideo_msg.h
MSG_F0_MPV_WRITE	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F0_PASS_ON	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F0_PIXEL_DEPTH_REQUEST	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F0_QUERY	/cig/include/mpvideo_msg.h; /cig/othersrc/force/force_rtsw_if.h, mpvideo_msg.h
MSG_F0_SET_DISPLAY	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F1_ACKNOWLEDGE	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F1_FINAL_LUT_ADDR	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F1_MPV_IOCTL	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F1_MPV_LUT_TYPE	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F1_MPV_MEMORY	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F1_PIXEL_ADDR	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F1_PIXEL_DEPTH_RETURN	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F1_STATUS	/cig/include/mpvideo_msg.h; /cig/othersrc/force/mpvideo_msg.h
MSG_F1_TEXT	/cig/include/mpvideo_msg.h; /cig/othersrc/force/force_rtsw_if.h, mpvideo_msg.h
MSG_FILE_DESCR	/cig/include/if_tst_ctl.h, sysdefs2.h
MSG_FILE_STATUS	/cig/include/if_tst_ctl.h, sysdefs2.h
MSG_FILE_XFER	/cig/include/if_tst_ctl.h, sysdefs2.h
MSG_GUN_OVERLAY	/cig/include/if_sim2cig.h
MSG_HDR	/cig/include/if_ctl_err.h
MSG_HIT_RETURN	/cig/include/if_cig2sim.h
MSG_HIT_RETURN48	/cig/include/if_rva2net.h
MSG_HPRXYZS_MATRIX	/cig/include/if_sim2cig.h
MSG_LASER_RETURN	/cig/include/if_cig2sim.h

<b>Data Type</b>	<b>Typedef Location</b>
MSG_LOCAL_TERRAIN	/cig/include/if_cig2sim.h
MSG_LT_PIECE	/cig/include/if_cig2sim.h
MSG_LT_STATE	/cig/include/if_sim2cig.h
MSG_MISS	/cig/include/if_cig2sim.h
MSG_OBSCURE	/cig/include/if_sim2cig.h
MSG_OTHERVEH_STATE	/cig/include/if_sim2cig.h
MSG_OVERLAY_SETUP	/cig/libsrc/libvpt/vpi_msgs.h
MSG_PASS_BACK	/cig/include/if_cig2sim.h
MSG_PASS_ON	/cig/include/if_sim2cig.h
MSG_PPM_DISPLAY_MODE	/cig/include/if_sim2cig.h
MSG_PPM_DISPLAY_OFFSET	/cig/include/if_sim2cig.h
MSG_PPM_PIXEL_LOCATION	/cig/include/if_sim2cig.h
MSG_PPM_PIXEL_STATE	/cig/include/if_sim2cig.h
MSG_PROCESS_CHORD	/cig/include/if_sim2cig.h
MSG_PROCESS_ROUND	/cig/include/if_sim2cig.h
MSG_PROCESS_ROUND48	/cig/include/if_sim2cig.h
MSG_REQUEST_LASER_RANGE	/cig/include/if_sim2cig.h
MSG_REQUEST_POINT_INFO	/cig/include/if_sim2cig.h
MSG_RETURN_POINT_INFO	/cig/include/if_cig2sim.h
MSG_ROT2x1_MATRIX	/cig/include/if_sim2cig.h
MSG_ROUND_FIRED	/cig/include/if_sim2cig.h
MSG_RTS4x3_MATRIX	/cig/include/if_sim2cig.h
MSG_SCALE	/cig/include/if_sim2cig.h
MSG_SHOT_REPORT	/cig/include/if_cig2sim.h
MSG_SHOT_REPORT48	/cig/include/if_rva2net.h
MSG_SHOW_EFFECT	/cig/include/if_sim2cig.h
MSG_SIO_CLOSE	/cig/include/if_sim2cig.h
MSG_SIO_INIT	/cig/include/if_sim2cig.h
MSG_SIO_WRITE	/cig/include/if_sim2cig.h
MSG_STATICVEH_REM	/cig/include/if_sim2cig.h
MSG_STATICVEH_STATE	/cig/include/if_sim2cig.h
MSG_STRUCT	/cig/include/cigsimio.h
MSG_SUBSYS_MODE	/cig/include/if_sim2cig.h
MSG_SYS_ERROR	/cig/include/if_ctl_err.h
MSG_TERRAIN_FEEDBACK_POINT_INIT	/cig/libsrc/libvpt/vpi_msgs.h
MSG_TERRAIN_FEEDBACK_SETUP	/cig/libsrc/libvpt/vpi_msgs.h
MSG_TEST_NAME	/cig/include/if_tst_ctl.h
MSG_TF_HDR	/cig/include/if_cig2sim.h
MSG_TF_INIT_HDR	/cig/include/if_sim2cig.h
MSG_TF_INIT_PT	/cig/include/if_sim2cig.h
MSG_TF_PT	/cig/include/if_cig2sim.h
MSG_TF_PT48	/cig/include/if_rva2net.h

<b>Data Type</b>	<b>Typedef Location</b>
MSG_TF_STATE	/cig/include/if_sim2cig.h
MSG_TF_VEHICLE_POS	/cig/include/if_sim2cig.h
MSG_TRAJ_CHORD	/cig/include/if_sim2cig.h
MSG_TRAJ_ENTRY	/cig/include/if_sim2cig.h
MSG_TRAJ_ENTRY_XFER	/cig/include/if_init.h
MSG_TRAJ_TABLE_XFER	/cig/include/if_init.h
MSG_TRANSLATION	/cig/include/if_sim2cig.h
MSG_VIEW_FLAGS	/cig/include/if_sim2cig.h
MSG_VIEW_MAGNIFICATION	/cig/include/if_sim2cig.h
MSG_VIEWPORT_STATE	/cig/include/if_init.h
MSG_VIEWPORT_STATE	/cig/libsrc/libvpt/vpi_msgs.h
MSG_VIEWPORT_UPDATE	/cig/include/if_sim2cig.h
MSGGS_BLK	/cig/include/if_ctl_err.h
MTXUNION	/cig/include/if_hdr_str.h, vpi_struct.h
MX2_DEVICE	/cig/libsrc/libgossip/mx3_hword.c; /cig/libsrc/librti/mx2_hword.c; /cig/othersrc/force/mx2_hword.c
MX_DEVICE	/cig/include/mx_defines.h
OMODEL	/cig/include/model_struct.h, structures.h
OVERLAY_PARAMS	/cig/include/overlay3d_struct.h
POI	/cig/include/demo_struct.h
POINT_DATA	/cig/include/bx_structs.h
POLY_CACHE_ENTRY	/cig/include/bx_structs.h
POLY_ENTRY	/cig/include/bx_structs.h
POLY_INFO_WORD	/cig/include/bx_rtdb_structs.h, poly_struct.h, structures.h
POLYGON_LIST	/cig/include/bx_rtdb_structs.h, poly_struct.h, structures.h
PPM_DATA_TYPE	/cig/include/ppm.h
PREAMBLE	/cig/libsrc/libhost/host_enet_if.c
PRINT_MSG_STRUCT	/cig/libsrc/libhost/host_if_debug.c; /cig/libsrc/libmsg/print_msg.c
PROJ_DATA	/cig/include/bal_struct.h, structures.h
PROJ_DATA_2	/cig/include/bal_struct.h, structures.h
R4BBOX2D	/cig/include/dgi_stdg.h, standard.h
R4BBOX3D	/cig/include/dgi_stdg.h, standard.h
R4HSL	/cig/include/dgi_stdg.h, standard.h
R4HSLO	/cig/include/dgi_stdg.h, standard.h
R4IJ	/cig/include/standard.h
R4MTX4X3	/cig/include/dgi_stdg.h, standard.h
R4MTX4X4	/cig/include/dgi_stdg.h, standard.h
R4P2D	/cig/include/dgi_stdg.h, standard.h
R4P3D	/cig/include/dgi_stdg.h, rt_types.h, standard.h
R4P4D	/cig/include/dgi_stdg.h, standard.h
R4RGB	/cig/include/dgi_stdg.h, standard.h

<b>Data Type</b>	<b>Typedef Location</b>
R4RGBO	/cig/include/dgi_stdg.h, standard.h
R8BBOX2D	/cig/include/dgi_stdg.h, standard.h
R8BBOX3D	/cig/include/dgi_stdg.h, standard.h
R8HSL	/cig/include/dgi_stdg.h, standard.h
R8HSLO	/cig/include/dgi_stdg.h, standard.h
R8MTX4X3	/cig/include/dgi_stdg.h, standard.h
R8MTX4X4	/cig/include/dgi_stdg.h, standard.h
R8P2D	/cig/include/dgi_stdg.h, standard.h
R8P3D	/cig/include/dgi_stdg.h, standard.h
R8P4D	/cig/include/dgi_stdg.h, standard.h
R8RGB	/cig/include/dgi_stdg.h, standard.h
R8RGBO	/cig/include/dgi_stdg.h, standard.h
RCL_UNION	/cig/include/rcinclude.h
REAL_4	/cig/include/dgi_stdh.h, rt_types.h, standard.h; /cig/othersrc/force/dgi_stdh.h
REAL_8	/cig/include/dgi_stdh.h, rt_types.h, standard.h; /cig/othersrc/force/dgi_stdh.h
RESOLUTION	/cig/include/if_hdr_str.h; /cig/libsrc/libvpt/vpi_msgs.h
RGBPOLY_LIST	/cig/include/poly_struct.h, structures.h
ROT2x1_MTX	/cig/include/if_hdr_str.h, vpi_struct.h
ROUND_DATA	/cig/include/bx_structs.h
RTS3x3_MTX	/cig/include/if_hdr_str.h, vpi_struct.h
RTS4x3_MTX	/cig/include/if_hdr_str.h, vpi_struct.h
SCREEN	/cig/include/vpi_viewport.h
SCREEN_POSITION_STRUCT	/cig/libsrc/libflea/tick_ppm.c
SCRN_CONSTANTS	/cig/include/vpi_viewport.h
SEARCH_LIST	/cig/include/definitions.h
SG_OVERLAY_SETUP	/cig/include/if_init.h
SHORT_LONG_ADDR	/cig/libsrc/libmpvideo/bootforce.c
SHOW_EFF	/cig/include/model_struct.h, structures.h
SIM_VEH_STRUCT	/cig/include/demo_struct.h
SIO_CTL	/cig/libsrc/libvio/sio.c
SOMODEL	/cig/include/model_struct.h, structures.h
SREM	/cig/include/model_struct.h, structures.h
STAMP_LIST	/cig/include/poly_struct.h, structures.h
STANK	/cig/include/model_struct.h, structures.h
STAT_VEH	/cig/include/bx_structs.h
STRING	/cig/include/standard.h, dgi_stdh.h; /cig/othersrc/force/dgi_stdh.h
STRUCT2D	/cig/include/struct_2d.h
STRUCT_P_BVOL	/cig/include/bx_structs.h
STRUCT_P_POLY	/cig/include/bx_structs.h
STRUCT_P_ROUND	/cig/include/bx_structs.h

<b>Data Type</b>	<b>Typedef Location</b>
STRUCT_P_SV	/cig/include/bx_structs.h
STRUCT_P_TF_PT	/cig/libsrc/libball/bx_tf_pack.c
SWITCH_2DLUT_STRUCT	/cig/othersrc/force/fl_defines.h
T_MATRIX[3][3]	/cig/include/vpi_struct.h
T_VECTOR[3]	/cig/include/vpi_struct.h
TABLE1	/cig/libsrc/libflea/flea_agpt_switches.c
TAC_STATUS	/cig/include/definitions.h
TANK	/cig/include/model_struct.h, structures.h
tasks	/cig/include/sysdefs2.h
TERRAIN_CORNERS	/cig/include/bx_structs.h
TEXTURE_INDEX	/cig/include/structures.h, traverse_cmd_defs.h
TEXTURE_MAP	/cig/include/poly_struct.h, structures.h
TF1	/cig/include/if_hdr_str.h
TF2	/cig/include/if_hdr_str.h
TF_DATA_HDR	/cig/include/bx_structs.h
TF_PT	/cig/libsrc/libball/bx_tf_pack.c
TF_TYPE	/cig/include/if_hdr_str.h; /cig/libsrc/libvpt/vpi_msgs.h
TRAJ_DATA	/cig/include/bal_struct.h, structures.h
TRAJ_DATA_2	/cig/include/bal_struct.h, structures.h
TRAJ_ENTRY	/cig/include/bx_structs.h
TRAJ_POS	/cig/include/bal_struct.h, structures.h
TRAJ_POS_2	/cig/include/bal_struct.h, structures.h
TRAJ_TABLE	/cig/include/bx_structs.h
UIR4P	/cig/include/standard.h, structures.h
UIR4P3D	/cig/include/standard.h, structures.h
UNS_1	/cig/include/dgi_std.h, /standard.h; /cig/othersrc/force/dgi_std.h
UNS_2	/cig/include/dgi_std.h, standard.h; /cig/othersrc/force/dgi_std.h
UNS_4	/cig/include/dgi_std.h, standard.h; /cig/othersrc/force/dgi_std.h
V_FLAGS	/cig/include/vpi_viewport.h
VECTOR4[3]	/cig/include/rt_types.h
VECTOR8[3]	/cig/include/rt_types.h
VIDEO_CONTROL	/cig/include/mpv_struct.h, structures.h
VIEWPORT_PARAMETERS	/cig/include/vpi_viewport.h
VPPOS_ARRAY	/cig/include/vpi_viewport.h
VPT_CNODE_INFO	/cig/include/vpi_query.h
VPT_PATH_INFO	/cig/include/vpi_query.h
VPT_VPT_INFO	/cig/include/vpi_query.h
WHERE_PROCESS	/cig/include/ecompile1.h
WINDOW_DESCRIPTOR_TABLE	/cig/include/struct_2d.h

**Data Type****Typedef Location**

WORD

`/cig/include/dgi_std.h, rt_types.h, standard.h;  
/cig/othersrc/force/dgi_std.h`

X'ZHPR

`/cig/include/if_hdr_str.h`

### E.3 Function Names Mapped To Source File Locations

The following list shows each function in the real-time software, and identifies the file in which the function is located. The third column shows the section number in which the function is described in this document.

Function Name	Location	Section
_copy_reconfigurable_viewports_section	/cig/libsrc/libmsg/msg_end.c	2.12.6.13
_database_disable	/cig/libsrc/libmsg/msg_end.c	2.12.6.10
_display_lights	/cig/libsrc/libmsg/msg_end.c	2.12.6.3
_downcount_effects	/cig/libsrc/libmsg/msg_end.c	2.12.6.2
_handle_point_lights	/cig/libsrc/libmsg/msg_end.c	2.12.6.11
_handle_request_local_terrain	/cig/libsrc/libmsg/msg_end.c	2.12.6.9
_move_load_module_stp_to_quad_buffer	/cig/libsrc/libmsg/msg_end.c	2.12.6.4
_pend_on_frame_interrupt	/cig/libsrc/libmsg/msg_end.c	2.12.6.6
_process_agl	/cig/libsrc/libmsg/msg_end.c	2.12.6.7
_reset_model_pointers	/cig/libsrc/libmsg/msg_end.c	2.12.6.12
_rowcol_rd	/cig/libsrc/librt/rowcol_rd.c	2.13.40.5
_rowcol_rd_cleanup	/cig/libsrc/librt/rowcol_rd.c	2.13.40.10
_set_up_for_next_frame	/cig/libsrc/libmsg/msg_end.c	2.12.6.8
_update_second_active_area_memory	/cig/libsrc/libmsg/msg_end.c	2.12.6.5
aam_free	/cig/libsrc/libvpt/be_stubs.c	2.17.1.9
aam_malloc	/cig/libsrc/libconfig/aam_manager.c	2.5.1.1
active_area_init	/cig/libsrc/libbackend/aa_init.c	2.3.1.1
addr_in_pool	/cig/libsrc/librt/ded_gm_pool.c	2.13.15.3
agpt_init	/cig/libsrc/librt/agpt_init.c	2.13.1
agpt_statistics	/cig/libsrc/libgossip/	2.9.1
autopilot	/cig/libsrc/libflea/autopilot.c	2.7.1
b0_aam_centroid	/cig/libsrc/libball/b0_aam_centroid.c	2.4.2.1
b0_aam_sw_corner	/cig/libsrc/libball/b0_aam_sw_corner.c	2.4.2.2
b0_add_static_vehicle	/cig/libsrc/libball/b0_add_static_vehicle.c	2.4.2.3
b0_add_traj_table	/cig/libsrc/libball/b0_add_traj_table.c	2.4.2.4
b0_bal_config	/cig/libsrc/libball/b0_bal_config.c	2.4.2.5
b0_bvol_entry	/cig/libsrc/libball/b0_bvol_entry.c	2.4.2.6
b0_cancel_round	/cig/libsrc/libball/b0_cancel_round.c	2.4.2.7
b0_cig_frame_rate	/cig/libsrc/libball/b0_cig_frame_rate.c	2.4.2.8
b0_database_info	/cig/libsrc/libball/b0_database_info.c	2.4.2.9
b0_delete_static_vehicle	/cig/libsrc/libball/b0_delete_static_vehicle.c	2.4.2.10
b0_delete_traj_table	/cig/libsrc/libball/b0_delete_traj_table.c	2.4.2.11
b0_error_detected	/cig/libsrc/libball/b0_error_detected.c	2.4.2.12
b0_inapp_message	/cig/libsrc/libball/b0_inapp_message.c	2.4.2.13
b0_lm_read	/cig/libsrc/libball/b0_lm_read.c	2.4.2.14
b0_model_directory	/cig/libsrc/libball/b0_model_directory.c	2.4.2.15
b0_model_entry	/cig/libsrc/libball/b0_model_entry.c	2.4.2.16
b0_new_frame	/cig/libsrc/libball/b0_new_frame.c	2.4.2.17
b0_print	/cig/libsrc/libball/b0_print.c	2.4.2.18
b0_process_chord	/cig/libsrc/libball/b0_process_chord.c	2.4.2.19
b0_process_round	/cig/libsrc/libball/b0_process_round.c	2.4.2.20
b0_round_fired	/cig/libsrc/libball/b0_round_fired.c	2.4.2.21
b0_state_control	/cig/libsrc/libball/b0_state_control.c	2.4.2.22
b0_status_request	/cig/libsrc/libball/b0_status_request.c	2.4.2.23
b0_tf_init_hdr	/cig/libsrc/libball/b0_tf_init_hdr.c	2.4.2.24
b0_tf_init_pt	/cig/libsrc/libball/b0_tf_init_pt.c	2.4.2.25
b0_tf_state	/cig/libsrc/libball/b0_tf_state.c	2.4.2.26
b0_tf_vehicle_pos	/cig/libsrc/libball/b0_tf_vehicle_pos.c	2.4.2.27



Function Name	Location	Section
b0_traj_chord	/cig/libsrc/libball/b0_traj_chord.c	2.4.2.28
b0_traj_entry	/cig/libsrc/libball/b0_traj_entry.c	2.4.2.29
b0_undefined_message	/cig/libsrc/libball/b0_undefined_message.c	2.4.2.30
backend_clear_laser_requests	/cig/libsrc/libbackend/backend_man.c	2.3.5.6
backend_get_object_addr	/cig/libsrc/libbackend/backend_man.c	2.3.5.5
backend_laser_request_range	/cig/libsrc/libbackend/backend_laser.c	2.3.4.1
backend_reset	/cig/libsrc/libbackend/backend_man.c	2.3.5.2
backend_response	/cig/libsrc/libbackend/backend_laser.c	2.3.4.2
backend_send_req	/cig/libsrc/libbackend/backend_man.c	2.3.5.4
backend_set_branch	/cig/libsrc/libbackend/backend_branch.c	2.3.2
backend_set_color	/cig/libsrc/libbackend/backend_color.c	2.3.3
backend_set_paths	/cig/libsrc/libbackend/backend_paths.c	2.3.6.1
backend_set_thermal	/cig/libsrc/libbackend/backend_thermal.c	2.3.7
backend_set_video	/cig/libsrc/libbackend/backend_video.c	2.3.8
backend_setup	/cig/libsrc/libbackend/backend_man.c	2.3.5.1
backend_sim_init	/cig/libsrc/libbackend/backend_man.c	2.3.5.3
backend_update_view_paths	/cig/libsrc/libbackend/backend_paths.c	2.3.6.2
bal_buffer_setup	/cig/libsrc/librtt/config_ballistics.c	2.13.10.2
bal_get_db_pos	/cig/libsrc/librtt/bal_get_db_pos.c	2.13.2
bal_get_lm_grid	/cig/libsrc/librtt/bal_get_lm_grid.c	2.13.3
ball_effect_add	/cig/libsrc/librtt/ball_effect_add.c	2.13.5
bbnctype	/cig/libsrc/libconfig/bbnctype.c	2.5.2
be_query_buffer_offset	/cig/libsrc/libvpt/be_stubs.c	2.17.1.3
be_query_db0	/cig/libsrc/libvpt/be_stubs.c	2.17.1.4
be_query_lm_per_lmb_side	/cig/libsrc/libvpt/be_stubs.c	2.17.1.6
be_query_num_paths	/cig/libsrc/libvpt/be_stubs.c	2.17.1.1
be_qulm	/cig/libsrc/libvpt/be_stubs.c	2.17.1.5
blank	/cig/libsrc/libutil/vt100.c	2.16.7.6
blcopy	/cig/libsrc/libutil/blcopy.c	2.16.1
bootforce	/cig/libsrc/libmpvideo/bootforce.c	2.11.1.1
bx_bvol_int	/cig/libsrc/libball/bx_bvol_int.c	2.4.3.1
bx_chord_intersect	/cig/libsrc/libball/bx_chord_intersect.c	2.4.3.2
bx_delete_round	/cig/libsrc/libball/bx_functions.c	2.4.3.5.2
bx_delete_stat_veh	/cig/libsrc/libball/bx_functions.c	2.4.3.5.10
bx_dist_sq_pt_line	/cig/libsrc/libball/bx_functions.c	2.4.3.5.11
bx_find_round_hit	/cig/libsrc/libball/bx_compute_round.c	2.4.3.3.5
bx_find_shot_report	/cig/libsrc/libball/bx_compute_round.c	2.4.3.3.3
bx_find_vehicle	/cig/libsrc/libball/bx_find_vehicle.c	2.4.3.4
bx_free_lm_cache	/cig/libsrc/libball/bx_functions.c	2.4.3.5.6
bx_get_chord_end	/cig/libsrc/libball/bx_functions.c	2.4.3.5.4
bx_get_db_pos	/cig/libsrc/libball/bx_functions.c	2.4.3.5.3
bx_get_lb_from_lm	/cig/libsrc/libball/bx_functions.c	2.4.3.5.8
bx_get_lm_data	/cig/libsrc/libball/bx_get_lm_data.c	2.4.3.6
bx_get_lm_grid	/cig/libsrc/libball/bx_get_lm_grid.c	2.4.3.7
bx_guntip_within_db	/cig/libsrc/libball/bx_compute_round.c	2.4.3.3.2
bx_init	/cig/libsrc/libball/bx_init.c	2.4.1.1
bx_model_int	/cig/libsrc/libball/bx_model_int.c	2.4.3.8
bx_new_bvol	/cig/libsrc/libball/bx_functions.c	2.4.3.5.5
bx_new_poly	/cig/libsrc/libball/bx_functions.c	2.4.3.5.7
bx_new_round	/cig/libsrc/libball/bx_functions.c	2.4.3.5.1
bx_new_stat_veh	/cig/libsrc/libball/bx_functions.c	2.4.3.5.9
bx_poly_int	/cig/libsrc/libball/bx_poly_int.c	2.4.3.9
bx_probe	/cig/libsrc/libball/bx_probe.c	2.4.1.2
bx_reset	/cig/libsrc/libball/bx_reset.c	2.4.1.3
bx_return_miss	/cig/libsrc/libball/bx_compute_round.c	2.4.3.3.1
bx_round_tracer_position	/cig/libsrc/libball/bx_compute_round.c	2.4.3.3.4
bx_task	/cig/libsrc/libball/bx_task.c	2.4.1.4.1
bx_task_cleanup	/cig/libsrc/libball/bx_task.c	2.4.1.4.2

Function Name	Location	Section
bx_tf_copy_msg	/cig/libsrc/libball/bx_tf_pack.c	2.4.3.10.4
bx_tf_free_tf_pts	/cig/libsrc/libball/bx_tf_pack.c	2.4.3.10.7
bx_tf_init_pt_cache	/cig/libsrc/libball/bx_tf_pack.c	2.4.3.10.1
bx_tf_new_tf_pts	/cig/libsrc/libball/bx_tf_pack.c	2.4.3.10.6
bx_tf_next	/cig/libsrc/libball/bx_tf_pack.c	2.4.3.10.3
bx_tf_pt_data	/cig/libsrc/libball/bx_tf_pack.c	2.4.3.10.5
bx_tf_pts	/cig/libsrc/libball/bx_tf_pack.c	2.4.3.10.2
bx_trajectory	/cig/libsrc/libball/bx_trajectory.c	2.4.3.11
cal	/cig/libsrc/librtt/cal.c	2.13.6
cancel_round	/cig/libsrc/libflea/encode_routines.c	2.7.4.28
check_restart	/cig/gtbsrc/rtt/rtt.c	2.1.3.2
check_sum	/cig/libsrc/libutil/sload.c	2.16.5.4
cig_2d_setup	/cig/libsrc/lib2d/cig_2d_setup.c	2.2.2
cig_config	/cig/libsrc/libconfig/cig_config.c	2.5.3
cigsimio_buffer_init	/cig/libsrc/librtt/cigsimio_obj.c	2.13.7.6
cigsimio_frame_end	/cig/libsrc/librtt/cigsimio_obj.c	2.13.7.7
cigsimio_get_data	/cig/libsrc/librtt/cigsimio_obj.c	2.13.7.5
cigsimio_msg_in	/cig/libsrc/librtt/cigsimio_obj.c	2.13.7.2
cigsimio_msg_out	/cig/libsrc/librtt/cigsimio_obj.c	2.13.7.3
cigsimio_write	/cig/libsrc/librtt/cigsimio_obj.c	2.13.7.4
clear	/cig/libsrc/libbackend/aa_init.c	2.3.1.2
clear_line	/cig/libsrc/libhost/host_if_debug.c	2.10.4.5
close_db	/cig/libsrc/librtt/close_db.c	2.13.8
CloseDir	/cig/libsrc/libutil/directory.c	2.16.2.3
cloud_init	/cig/libsrc/librtt/clouds.c	2.13.9.1
cloud_mgmt	/cig/libsrc/librtt/clouds.c	2.13.9.4
cloud_placement	/cig/libsrc/librtt/clouds.c	2.13.9.5
cloud_scud	/cig/libsrc/librtt/clouds.c	2.13.9.6
cloud_update	/cig/libsrc/librtt/clouds.c	2.13.9.2
cloud_update_model	/cig/libsrc/librtt/clouds.c	2.13.9.3
compare_buffers	/cig/othersrc/force/forcetask.c	2.18.32.2
compile_2d (CIG version)	/cig/libsrc/lib2d/cig_comp_2d.c	2.2.3
compile_2d (offline version)	/cig/libsrc/lib2d/u_comp_2d.c	2.2.15
concat_mtx	/cig/libsrc/libvpt/mtx_concat.c	2.17.11
config_ballistics	/cig/libsrc/librtt/config_ballistics.c	2.13.10.1
config_color_table	/cig/libsrc/librtt/config_color_table.c	2.13.11
config_database	/cig/libsrc/librtt/config_database.c	2.13.12.1
config_translator	/cig/libsrc/libflea/cf_translator.c	2.7.2.1
ctoi	/cig/libsrc/libutil/sload.c	2.16.5.7
cup	/cig/libsrc/libutil/vt100.c	2.16.7.1
data_type	/cig/othersrc/force/data_type.c	2.18.1
db_mcc_setup	/cig/libsrc/librtt/db_mcc_setup.c	2.13.13
debug_initdr	/cig/libsrc/librtt/debug_initdr.c	2.13.14
ded_add_effect	/cig/libsrc/libgossip/ded_object.c	2.9.3.4
ded_add_model	/cig/libsrc/libgossip/ded_object.c	2.9.3.3
ded_adjust_addr_tables	/cig/libsrc/libgossip/ded_object.c	2.9.3.6
ded_cross_border	/cig/libsrc/libgossip/ded_object.c	2.9.3.9
ded_dtp_trace	/cig/libsrc/libgossip/ded_object.c	2.9.3.12
ded_init_mdl_addr	/cig/libsrc/libgossip/ded_object.c	2.9.3.2
ded_load_dtp_code	/cig/libsrc/libgossip/ded_object.c	2.9.3.13
ded_model_end_addr	/cig/libsrc/libgossip/ded_object.c	2.9.3.5
ded_model_offset	/cig/libsrc/libgossip/ded_object.c	2.9.3.14
ded_model_trace	/cig/libsrc/librtt/ded_model_trace.c	2.13.16
ded_object_debug	/cig/libsrc/libgossip/ded_object.c	2.9.3.16
ded_print_tables	/cig/libsrc/libgossip/ded_object.c	2.9.3.15
ded_process_directory	/cig/libsrc/libgossip/ded_object.c	2.9.3.11
ded_relocate_model	/cig/libsrc/libgossip/ded_object.c	2.9.3.10
ded_setup	/cig/libsrc/libgossip/ded_object.c	2.9.3.1

Function Name	Location	Section
ded_sub_end	/cig/libsrc/libgossip/ded_object.c	2.9.3.8
ded_uninit	/cig/libsrc/libgossip/ded_object.c	2.9.3.7
derror	/cig/libsrc/libflea/flea_agpt_switches.c	2.7.8.7
disable_restart	/cig/gtbinsrc/rtu/rtu.c	2.1.3.3
display	/cig/libsrc/libgossip/dtp_emu.c	2.9.4.2
dl_setup	/cig/libsrc/libbackend/dl_man.c	2.3.9
dl_setup	/cig/libsrc/librtt/dl_man.c	2.13.17
double_bot	/cig/libsrc/libutil/vt100.c	2.16.7.4
double_lite	/cig/libsrc/libgossip/replace_mod.c	2.9.18.3
double_off	/cig/libsrc/libutil/vt100.c	2.16.7.5
double_top	/cig/libsrc/libutil/vt100.c	2.16.7.3
download_bvols	/cig/libsrc/librtt/download_bvols.c	2.13.18
dtp_compiler	/cig/libsrc/libgendtp/dtp_compiler.c	2.8.1
dtp_emu	/cig/libsrc/libgossip/dtp_emu.c	2.9.4.1
dtp_malloc	/cig/libsrc/libgendtp/dtp_funcs.c	2.8.2.5
dtp_malloc_init	/cig/libsrc/libgendtp/dtp_funcs.c	2.8.2.6
dtp_trav1	/cig/libsrc/libgendtp/dtp_trav1.c	2.8.3
dtp_trav2	/cig/libsrc/libgendtp/dtp_trav2.c	2.8.4
dynamic_aam_init	/cig/libsrc/libconfig/aam_manager.c	2.5.1.4
dynamic_demo	/cig/libsrc/libflea/dynamic_demo.c	2.7.3
effect_downcount	/cig/libsrc/librtt/effect_downcount.c	2.13.19
errors	/cig/libsrc/librtt/config_database.c	2.13.12.2
esifa_ConfigData	/cig/libsrc/libesifa/esifa_man.c	2.6.4.5
esifa_download	/cig/libsrc/libesifa/esifa_query.c	2.6.5.5
esifa_get_object_addr	/cig/libsrc/libesifa/esifa_man.c	2.6.4.4
esifa_laser_request_range	/cig/libsrc/libesifa/esifa_laser.c	2.6.2.1
esifa_laser_return	/cig/libsrc/libesifa/esifa_laser.c	2.6.2.2
esifa_load	/cig/libsrc/libesifa/esifa_load.c	2.6.3
esifa_queue_data	/cig/libsrc/libesifa/esifa_man.c	2.6.4.6
esifa_queue_download	/cig/libsrc/libesifa/esifa_man.c	2.6.4.7
esifa_read	/cig/libsrc/libesifa/esifa_query.c	2.6.5.3
esifa_read_ports	/cig/libsrc/libesifa/esifa_query.c	2.6.5.1
esifa_send_queue	/cig/libsrc/libesifa/esifa_man.c	2.6.4.8
esifa_send_req	/cig/libsrc/libesifa/esifa_man.c	2.6.4.3
esifa_set_fade	/cig/libsrc/libesifa/esifa_fade.c	2.6.1
esifa_set_special	/cig/libsrc/libesifa/esifa_special.c	2.6.6
esifa_set_thermal	/cig/libsrc/libesifa/esifa_thermal.c	2.6.7
esifa_set_video	/cig/libsrc/libesifa/esifa_video.c	2.6.8
esifa_setup	/cig/libsrc/libesifa/esifa_man.c	2.6.4.1
esifa_sim_init	/cig/libsrc/libesifa/esifa_man.c	2.6.4.2
esifa_write	/cig/libsrc/libesifa/esifa_query.c	2.6.5.4
esifa_write_ports	/cig/libsrc/libesifa/esifa_query.c	2.6.5.2
excep_init	/cig/other/src/force/exception.asm	2.18.2.1
exchange_dr11_data	/cig/libsrc/libhost/host_dr11_if.c	2.10.1.2
exchange_dr11_data_sim	/cig/libsrc/libhost/host_dr11_if.c	2.10.1.3
exchange_enet_data	/cig/libsrc/libhost/host_enet_if.c	2.10.2.2
exchange_enet_data_sim	/cig/libsrc/libhost/host_enet_if.c	2.10.2.3
exchange_flea_data	/cig/libsrc/libhost/host_flea_if.c	2.10.3.2
exchange_mpv_data	/cig/libsrc/libhost/host_mpv_if.c	2.10.5.2
exchange_mpv_data_sim	/cig/libsrc/libhost/host_mpv_if.c	2.10.5.3
exchange_scsi_data	/cig/libsrc/libhost/host_scsi_if.c	2.10.6.2
exchange_scsi_data_sim	/cig/libsrc/libhost/host_scsi_if.c	2.10.6.3
exchange_socket_data	/cig/libsrc/libhost/host_socket_if.c	2.10.7.2
exchange_socket_data_sim	/cig/libsrc/libhost/host_socket_if.c	2.10.7.3
extended_ram_available	/cig/libsrc/libbackend/aa_init.c	2.3.1.3
f0_3dlut_download	/cig/other/src/force/f0_3dlut_download.c	2.18.3
f0_3dlut_switch	/cig/other/src/force/f0_3dlut_switch.c	2.18.4
f0_alllut_switch	/cig/other/src/force/f0_alllut_switch.c	2.18.5

Function Name	Location	Section
f0_debug_disable	/cig/othersrc/force/f0_debug_disable.c	2.18.6
f0_debug_enable	/cig/othersrc/force/f0_debug_enable.c	2.18.7
f0_final_lut_download	/cig/othersrc/force/f0_final_lut_download.c	2.18.8
f0_final_lut_switch	/cig/othersrc/force/f0_final_lut_switch.c	2.18.9
f0_mode_select	/cig/othersrc/force/f0_mode_select.c	2.18.10
f0_mpv_init	/cig/othersrc/force/f0_mpv_init.c	2.18.11
f0_mpv_lut_type_request	/cig/othersrc/force/f0_mpv_lut_type_request.c	2.18.12
f0_mpv_peek	/cig/othersrc/force/f0_mpv_peek.c	2.18.13
f0_mpv_poke	/cig/othersrc/force/f0_mpv_poke.c	2.18.14
f0_mpv_poke16	/cig/othersrc/force/f0_mpv_poke16.c	2.18.15
f0_mpv_reset	/cig/othersrc/force/f0_mpv_reset.c	2.18.16
f0_mpv_task_control	/cig/othersrc/force/f0_mpv_task_control.c	2.18.17
f0_mpv_test	/cig/othersrc/force/f0_mpv_test.c	2.18.18
f0_mpv_write	/cig/othersrc/force/f0_mpv_write.c	2.18.19
f0_pass_on	/cig/othersrc/force/f0_pass_on.c	2.18.20
f0_pixel_depth_request	/cig/othersrc/force/f0_pixel_depth_request.c	2.18.21
f0_query	/cig/othersrc/force/f0_query.c	2.18.22.1
f0_set_display	/cig/othersrc/force/f0_set_display.c	2.18.23
f0_trigger	/cig/othersrc/force/f0_trigger.c	2.18.24
f0_unknown	/cig/othersrc/force/f0_unknown.c	2.18.25
f1_force_init	/cig/othersrc/force/f1_force_init.c	2.18.26
f1_init_jump_table	/cig/othersrc/force/f1_init_jump_table.c	2.18.27
f1_pa_320x240_h	/cig/othersrc/force/f1_pixel_address.c	2.18.28.10
f1_pa_320x240_v	/cig/othersrc/force/f1_pixel_address.c	2.18.28.6
f1_pa_640x240_h	/cig/othersrc/force/f1_pixel_address.c	2.18.28.12
f1_pa_640x240_v	/cig/othersrc/force/f1_pixel_address.c	2.18.28.8
f1_pa_640x256_h	/cig/othersrc/force/f1_pixel_address.c	2.18.28.11
f1_pa_640x256_v	/cig/othersrc/force/f1_pixel_address.c	2.18.28.7
f1_pa_640x480_h	/cig/othersrc/force/f1_pixel_address.c	2.18.28.9
f1_pa_640x480_v	/cig/othersrc/force/f1_pixel_address.c	2.18.28.5
f1_pa_fb_offset	/cig/othersrc/force/f1_pixel_address.c	2.18.28.4
f1_pa_init	/cig/othersrc/force/f1_pixel_address.c	2.18.28.1
f1_pa_new_orientation	/cig/othersrc/force/f1_pixel_address.c	2.18.28.2
f1_pa_new_resolution	/cig/othersrc/force/f1_pixel_address.c	2.18.28.3
f1_process_messages	/cig/othersrc/force/f1_process_messages.c	2.18.29
f1_setup_environment	/cig/othersrc/force/f1_setup_environment.c	2.18.30
file_control	/cig/libsrc/librtu/file_control.c	2.13.20
fill_tree	/cig/libsrc/libconfig/fill_tree.c	2.5.4.1
find_be_id	/cig/libsrc/libvpt/be_stubs.c	2.17.1.2
find_fn	/cig/libsrc/libutil/find_gfn.c	2.16.4
find_pitch_and_roll	/cig/libsrc/libflea/flea_simulate_vehicles.c	2.7.20.2
FindField	/cig/libsrc/libutil/find_field.c	2.16.3
fire_round	/cig/libsrc/libflea/encode_routines.c	2.7.4.25
flagoff	/cig/libsrc/libvpt/flagoff.c	2.17.7
flea	/cig/libsrc/libflea/flea.c	2.7.5.1
flea_abs_playback	/cig/libsrc/libflea/flea_script.c	2.7.19.1
flea_agl_terrain_follow	/cig/libsrc/libflea/flea_agl_terrain_follow.c	2.7.6
flea_agpt_locations	/cig/libsrc/libflea/flea_agpt_locations.c	2.7.7.1
flea_agpt_locations_main_menu	/cig/libsrc/libflea/flea_agpt_locations.c	2.7.7.2
flea_agpt_switches	/cig/libsrc/libflea/flea_agpt_switches.c	2.7.8.1
flea_agpt_switches_main_menu	/cig/libsrc/libflea/flea_agpt_switches.c	2.7.8.6
flea_atp	/cig/libsrc/libflea/flea_atp.c	2.7.9.1
flea_atp_main_menu	/cig/libsrc/libflea/flea_atp.c	2.7.9.2
flea_bal_opts	/cig/libsrc/libflea/flea_bal_opts.c	2.7.10.1
flea_bal_opts_main_menu	/cig/libsrc/libflea/flea_bal_opts.c	2.7.10.2
flea_calibration_image	/cig/libsrc/libflea/flea_ppm_obj.c	2.7.18.2
flea_cleanup	/cig/libsrc/libflea/flea.c	2.7.5.11
flea_db_traverse	/cig/libsrc/libflea/flea_db_traverse.c	2.7.11

Function Name	Location	Section
flea_decode_data	/cig/libsrc/libflea/flea_decode_data.c	2.7.12
flea_demo	/cig/libsrc/libflea/flea_demo.c	2.7.13
flea_draw_2d	/cig/libsrc/libflea/flea_draw_2d.c	2.7.14
flea_dummy_getchar	/cig/libsrc/libflea/flea.c	2.7.5.4
flea_encode_data	/cig/libsrc/libflea/flea_encode_data.c	2.7.15
flea_error_print	/cig/libsrc/libflea/encode_routines.c	2.7.4.22
flea_getchar	/cig/libsrc/libflea/flea.c	2.7.5.3
flea_graphics_test	/cig/libsrc/libflea/flea_graphics_test.c	2.7.16.1
flea_graphics_test_main_menu	/cig/libsrc/libflea/flea_graphics_test.c	2.7.16.2
flea_host_if		
flea_init_cig_sw	/cig/libsrc/libflea/flea_init_cig_sw.c	2.7.17
flea_initialized	/cig/libsrc/libflea/flea.c	2.7.5.8
flea_IO_mode	/cig/libsrc/libflea/flea.c	2.7.5.5
flea_IO_off	/cig/libsrc/libflea/flea.c	2.7.5.7
flea_IO_on	/cig/libsrc/libflea/flea.c	2.7.5.6
flea_io_task	/cig/libsrc/libflea/flea.c	2.7.5.2
flea_io_task_cleanup	/cig/libsrc/libflea/flea.c	2.7.5.12
flea_ppm_display_mode	/cig/libsrc/libflea/flea_ppm_obj.c	2.7.18.3
flea_ppm_display_offset	/cig/libsrc/libflea/flea_ppm_obj.c	2.7.18.4
flea_ppm_pixel_location	/cig/libsrc/libflea/flea_ppm_obj.c	2.7.18.5
flea_ppm_pixel_state	/cig/libsrc/libflea/flea_ppm_obj.c	2.7.18.6
flea_printf	/cig/libsrc/libflea/flea.c	2.7.5.10
flea_prompt	/cig/libsrc/libflea/tick.c	2.7.26.3
flea_simulate_vehicles	/cig/libsrc/libflea/flea_simulate_vehicles.c	2.7.20.1
flea_switches	/cig/libsrc/libflea/flea_switches.c	2.7.21.1
flea_switches_main_menu	/cig/libsrc/libflea/flea_switches.c	2.7.21.2
flea_update_pos	/cig/libsrc/libflea/flea_update_pos.c	2.7.22
flea_veh_control	/cig/libsrc/libflea/flea_veh_control.c	2.7.23.1
flea_veh_control_main_menu	/cig/libsrc/libflea/flea_veh_control.c	2.7.23.2
ftoh	/cig/libsrc/libgossip/dtp_emu.c	2.9.4.6
func_msg (config_database)	/cig/libsrc/librtu/config_database.c	2.13.12.3
func_msg (open_dbase)	/cig/libsrc/librtu/open_dbase.c	2.13.36.2
fxbvtofl	/cig/libsrc/librtu/fxbvtofl.c	2.13.21.2
fxbvtofl_020	/cig/libsrc/librtu/fxbvtofl.c	2.13.21.3
fxbvtofl_dart	/cig/libsrc/librtu/fxbvtofl.c	2.13.21.1
generic_lm	/cig/libsrc/librtu/generic_lm.c	2.13.22.2
get_3d_lut_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.6
get_binary_data	/cig/libsrc/libutil/sload.c	2.16.5.5
get_char	/cig/libsrc/libutil/sload.c	2.16.5.6
get_color_config_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.9
get_current_token	/cig/libsrc/libtoken/lex.c	2.15.1.8
get_data_2d_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.7
get_database_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.3
get_ded_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.4
get_dtp_bank	/cig/libsrc/librtu/sys_control.c	2.13.46.6
get_esifa_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.8
get_final_lut_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.5
get_lm	/cig/libsrc/libgossip/dtp_emu.c	2.9.4.9
get_msg_2d	/cig/libsrc/lib2d/cig_getm_2d.c	2.2.4
get_msg_2d (offline version)	/cig/libsrc/lib2d/u_getm_2d.c	2.2.16
get_next_message	/cig/libsrc/libflea/flea_script.c	2.7.19.3
get_next_packet	/cig/libsrc/libflea/flea_script.c	2.7.19.2
get_next_token	/cig/libsrc/libtoken/lex.c	2.15.1.7
get_number_value	/cig/libsrc/libtoken/lex.c	2.15.1.9
get_record	/cig/libsrc/libutil/sload.c	2.16.5.2
get_sio_write_data	/cig/libsrc/libflea/get_sio_data.c	2.7.24
get_string_value	/cig/libsrc/libtoken/lex.c	2.15.1.10
get_thing	/cig/libsrc/lib2d/get_thing.c	2.2.8

Function Name	Location	Section
get_tx_lut_index	/cig/libsrc/librtt/get_tx_lut_index.c	2.13.23
get_vehicle_position	/cig/libsrc/libconfig/get_vehicle_position.c	2.5.5
getch	/cig/libsrc/libconfig/getch.c	2.5.6
GetFileName	/cig/libsrc/libutil/directory.c	2.16.2.4
getlmdp	/cig/libsrc/librtt/load_modules.c	2.13.30.2
getmatrix	/cig/libsrc/librtt/mkmtx_nt.c	2.13.33.10
getside	/cig/libsrc/librtt/load_modules.c	2.13.30.3
gos_120tx	/cig/libsrc/libgossip/gos_120tx.c	2.9.5
gos_bal_query	/cig/libsrc/libgossip/gos_bal_query.c	2.9.6
gos_cigsimio	/cig/libsrc/librtt/cigsimio_obj.c	2.13.7.1
gos_db_query	/cig/libsrc/libgossip/gos_db_query.c	2.9.7.1
gos_db_query_menu	/cig/libsrc/libgossip/gos_db_query.c	2.9.7.3
gos_display_db_info	/cig/libsrc/libgossip/gos_db_query.c	2.9.7.2
gos_dummy_getchar	/cig/libsrc/libgossip/gossip.c	2.9.15.9
gos_getchar	/cig/libsrc/libgossip/gossip.c	2.9.15.8
gos_IO_off	/cig/libsrc/libgossip/gossip.c	2.9.15.11
gos_IO_on	/cig/libsrc/libgossip/gossip.c	2.9.15.10
gos_locate	/cig/libsrc/libgossip/gos_locate.c	2.9.8
gos_main_menu	/cig/libsrc/libgossip/gossip.c	2.9.15.7
gos_memory	/cig/libsrc/libgossip/gos_memory.c	2.9.9
gos_model	/cig/libsrc/libgossip/gos_model.c	2.9.10.1
gos_mpv	/cig/libsrc/libgossip/gos_mpv.c	2.9.11
gos_mpvio	/cig/libsrc/libgossip/gos_mpvio.c	2.9.12
gos_polys	/cig/libsrc/libgossip/gos_polys.c	2.9.13
gos_ppm_query	/cig/libsrc/libbackend/ppm_obj.c	2.3.10.5
gos_ppm_query_menu	/cig/libsrc/libbackend/ppm_obj.c	2.3.10.6
gos_prompt	/cig/libsrc/libgossip/gossip.c	2.9.15.6
gos_single_step	/cig/libsrc/libgossip/gossip.c	2.9.15.5
gos_system	/cig/libsrc/libgossip/gos_system.c	2.9.14
gos_timing_printout	/cig/libsrc/libgossip/gossip.c	2.9.15.3
gossip	/cig/libsrc/libgossip/gossip.c	2.9.15.1
gossip_cleanup	/cig/libsrc/libgossip/gossip.c	2.9.15.12
gossip_tick	/cig/libsrc/libgossip/gossip.c	2.9.15.2
gsp_io	/cig/othersrc/force/gsp_io.c	2.18.33
gsp_ioctl_read	/cig/othersrc/force/force.asm	2.18.31.4
gsp_ioctl_write	/cig/othersrc/force/force.asm	2.18.31.5
gsp_read	/cig/othersrc/force/force.asm	2.18.31.3
gsp_reset	/cig/othersrc/force/force.asm	2.18.31.6
gsp_write	/cig/othersrc/force/force.asm	2.18.31.2
hexdisplay	/cig/libsrc/libgossip/dtp_emu.c	2.9.4.5
host_disable_all_debug_msgs	/cig/libsrc/libhost/host_if_debug.c	2.10.4.3
host_enable_all_debug_msgs	/cig/libsrc/libhost/host_if_debug.c	2.10.4.2
host_if_buffer_init	/cig/libsrc/librtt/global_init.c	2.13.24.1
host_if_debug	/cig/libsrc/libhost/host_if_debug.c	2.10.4.13
host_if_debug_init	/cig/libsrc/libhost/host_if_debug.c	2.10.4.4
host_if_debug_main_menu	/cig/libsrc/libhost/host_if_debug.c	2.10.4.10
host_if_debug_menu	/cig/libsrc/libhost/host_if_debug.c	2.10.4.11
host_if_debug_tick	/cig/libsrc/libhost/host_if_debug.c	2.10.4.12
host_if_disable_debug_msgs	/cig/libsrc/libhost/host_if_debug.c	2.10.4.9
host_if_display_enabled_msgs	/cig/libsrc/libhost/host_if_debug.c	2.10.4.6
host_if_enable_debug_msgs	/cig/libsrc/libhost/host_if_debug.c	2.10.4.8
host_init_packet_sizes	/cig/libsrc/librtt/global_init.c	2.13.24.2
host_list_msgs	/cig/libsrc/libhost/host_if_debug.c	2.10.4.7
htof	/cig/libsrc/libgossip/dtp_emu.c	2.9.4.7
hw_test	/cig/libsrc/librtt/hw_test.c	2.13.26
hxflt	/cig/libsrc/libgossip/dtp_emu.c	2.9.4.4
id_4x3mtx	/cig/libsrc/librtt/mkmtx_nt.c	2.13.33.6
id_matrix	/cig/libsrc/librtt/make_bbn.c	2.13.32.6

Function Name	Location	Section
init_dr11_interface	/cig/libsrc/libhost/host_dr11_if.c	2.10.1.4
init_dtp_stacks	/cig/libsrc/libgendtp/dtp_funcs.c	2.8.2.4
init_enet_interface	/cig/libsrc/libhost/host_enet_if.c	2.10.2.5
init_flea_interface	/cig/libsrc/libhost/host_flea_if.c	2.10.3.4
init_generic_lm	/cig/libsrc/librtu/generic_lm.c	2.13.22.1
init_mpv_interface	/cig/libsrc/libhost/host_mpv_if.c	2.10.5.4
init_ports	/cig/other/src/force/force.asm	2.18.31.1
init_print_msg_array	/cig/libsrc/libmsg/print_msg.c	2.12.19.2
init_scsi_interface	/cig/libsrc/libhost/host_scsi_if.c	2.10.6.4
init_simulation	/cig/libsrc/librtu/init_sim.c	2.13.27
init_socket_interface	/cig/libsrc/libhost/host_socket_if.c	2.10.7.4
init_stuff	/cig/libsrc/lib2d/init_stuff.c	2.2.9
init_subsys_parser	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.1
initialize	/cig/libsrc/librtu/rtu_init.c	2.13.42.2
initialize_defaults	/cig/libsrc/librtu/rtu_init.c	2.13.42.1
linkup	/cig/libsrc/lib2d/cig_link_2d.c	2.2.5
linkup (offline version)	/cig/libsrc/lib2d/u_link_2d.c	2.2.17
load_dbase	/cig/libsrc/librtu/load_dbase.c	2.13.28
load_esifa	/cig/libsrc/librtu/load_esifa.c	2.13.29
load_modules	/cig/libsrc/librtu/load_modules.c	2.13.30.1
loc_ter_msg	/cig/libsrc/libmsg/loc_ter_msg.c	2.12.1
local_terrain	/cig/libsrc/librtu/loc_ter.c	2.13.31.1
local_terrain_cleanup	/cig/libsrc/librtu/loc_ter.c	2.13.31.2
lt_state	/cig/libsrc/libmsg/msg_lt_state.c	2.12.9
m1_gun_overlay	/cig/libsrc/librtu/gun_overlays.c	2.13.25.1
m2_gun_overlay	/cig/libsrc/librtu/gun_overlays.c	2.13.25.2
main (ballistics)	/cig/gtbinsrc/bx147_main.c	2.1.1.1
main (force)	/cig/other/src/force/forcetask.c	2.18.32.1
main (rtu)	/cig/gtbinsrc/rtu/rtu.c	2.1.3.1
main (2-D compiler)	/cig/libsrc/lib2d/u_main_2d.c	2.2.18
make4x3	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.21
make_bbn_logo	/cig/libsrc/libgossip/make_bbn_logo.c	2.9.16
make_cal_matrices	/cig/gtbinsrc/rtu/mkcal.c	2.1.2.2
make_cal_overlay	/cig/gtbinsrc/rtu/mkcal.c	2.1.2.1
make_cal_patterns	/cig/gtbinsrc/rtu/mkcal.c	2.1.2.3
make_m1_overlays	/cig/libsrc/librtu/gun_overlays.c	2.13.25.3
make_m2_overlays	/cig/libsrc/librtu/gun_overlays.c	2.13.25.4
make_p_nt	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.1
mat_adjugate	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.15
mat_copy	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.13
mat_determinant	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.18
mat_inverse	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.19
mat_mult	/cig/libsrc/libgossip/dtp_emu.c	2.9.4.8
mat_scale	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.17
mat_transpose	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.16
mat_vec_mul	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.14
matrix2	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.11
mem_check	/cig/libsrc/libvpt/tst_tree.c	2.17.22.2
menu_header	/cig/libsrc/libflea/tick.c	2.7.26.5
model_demo	/cig/libsrc/libflea/model_demo.c	2.7.25
model_mtx	/cig/libsrc/librtu/model_mtx.c	2.13.34
mpvideo_boot	/cig/libsrc/libmpvideo/bootmpv.c	2.11.2.1
mpvideo_define_mode	/cig/libsrc/libmpvideo/mpvideo_mode.c	2.11.7.2
mpvideo_get_object_addr	/cig/libsrc/libmpvideo/mpvideo_man.c	2.11.6.6
mpvideo_laser_request_range	/cig/libsrc/libmpvideo/mpvideo_laser.c	2.11.4
mpvideo_load	/cig/libsrc/libmpvideo/loadmpv.c	2.11.3
mpvideo_num_paths	/cig/libsrc/libmpvideo/mpvideo_query.c	2.11.11
mpvideo_pass_back	/cig/libsrc/libmpvideo/mpvideo_pass_back.c	2.11.8

Function Name	Location	Section
mpvideo_pass_on	/cig/libsrc/libmpvideo/mpvideo_pass_on.c	2.11.9
mpvideo_response	/cig/libsrc/libmpvideo/mpvideo_response.c	2.11.12
mpvideo_send_req	/cig/libsrc/libmpvideo/mpvideo_man.c	2.11.6.5
mpvideo_set_lut	/cig/libsrc/libmpvideo/mpvideo_lut.c	2.11.5
mpvideo_set_mode	/cig/libsrc/libmpvideo/mpvideo_mode.c	2.11.7.1
mpvideo_set_video	/cig/libsrc/libmpvideo/mpvideo_man.c	2.11.6.1
mpvideo_setup	/cig/libsrc/libmpvideo/mpvideo_man.c	2.11.6.2
mpvideo_sim_init	/cig/libsrc/libmpvideo/mpvideo_man.c	2.11.6.4
mpvideo_stop	/cig/libsrc/libmpvideo/mpvideo_man.c	2.11.6.3
mpvmsg_from_buf_addr	/cig/libsrc/libmpvideo/bootforce.c	2.11.1.7
mpvmsg_query_buf_addr	/cig/libsrc/libmpvideo/bootforce.c	2.11.1.4
mpvmsg_reply_buf_addr	/cig/libsrc/libmpvideo/bootforce.c	2.11.1.5
mpvmsg_to_buf_addr	/cig/libsrc/libmpvideo/bootforce.c	2.11.1.6
msg_lrotation	/cig/libsrc/libmsg/msg_vport.c	2.12.18.8
msg_3rotations	/cig/libsrc/libmsg/msg_vport.c	2.12.18.9
msg_calibration_image	/cig/libsrc/libmsg/msg_calibration_image.c	2.12.2
msg_cig_ctl	/cig/libsrc/libmsg/msg_cig_ctl.c	2.12.3
msg_dr11_pkt_size	/cig/libsrc/libmsg/msg_dr11.c	2.12.4
msg_end	/cig/libsrc/libmsg/msg_end.c	2.12.6.1
msg_hprxyzs_matrix	/cig/libsrc/libmsg/msg_vport.c	2.12.18.5
msg_laser_request_range	/cig/libsrc/libmsg/msg_laser.c	2.12.7
msg_laser_return	/cig/libsrc/libmsg/msg_laser_return.c	2.12.8
msg_otherveh_state	/cig/libsrc/libmsg/msg_veh_state.c	2.12.16.1
msg_pass_back	/cig/libsrc/libmsg/msg_pass_back.c	2.12.10
msg_pass_on	/cig/libsrc/libmsg/msg_pass_on.c	2.12.11
msg_ppm_display_mode	/cig/libsrc/libmsg/msg_ppm.c	2.12.12.1
msg_ppm_display_offset	/cig/libsrc/libmsg/msg_ppm.c	2.12.12.2
msg_ppm_pixel_location	/cig/libsrc/libmsg/msg_ppm.c	2.12.12.3
msg_ppm_pixel_state	/cig/libsrc/libmsg/msg_ppm.c	2.12.12.4
msg_process_round48	/cig/libsrc/libmsg/msg_process_round48.c	2.12.13
msg_rot2x1_matrix	/cig/libsrc/libmsg/msg_vport.c	2.12.18.3
msg_rts4x3_matrix	/cig/libsrc/libmsg/msg_vport.c	2.12.18.4
msg_scale	/cig/libsrc/libmsg/msg_vport.c	2.12.18.7
msg_shell_sort	/cig/libsrc/libhost/host_if_debug.c	2.10.4.1
msg_show_effect	/cig/libsrc/libmsg/msg_effect.c	2.12.5
msg_staticveh_rem	/cig/libsrc/libmsg/msg_veh_state.c	2.12.16.3
msg_staticveh_state	/cig/libsrc/libmsg/msg_veh_state.c	2.12.16.2
msg_subsys_mode	/cig/libsrc/libmsg/msg_subsys_mode.c	2.12.14
msg_syserr	/cig/libsrc/libmsg/msg_syserr.c	2.12.15
msg_translation	/cig/libsrc/libmsg/msg_vport.c	2.12.18.6
msg_view_flags	/cig/libsrc/libmsg/msg_vflags.c	2.12.17
msg_view_magnification	/cig/libsrc/libmsg/msg_vport.c	2.12.18.2
msg_viewport_update	/cig/libsrc/libmsg/msg_vport.c	2.12.18.1
mtx_non_perspective	/cig/libsrc/libvpt/mtx_viewspace.c	2.17.13.1
mtx_perspective	/cig/libsrc/libvpt/mtx_viewspace.c	2.17.13.2
mtxcpy	/cig/libsrc/librtt/mkmtx_nt.c	2.13.33.12
mult_4x3mtx	/cig/libsrc/librtt/mkmtx_nt.c	2.13.33.9
multmatrix	/cig/libsrc/librtt/make_bbn.c	2.13.32.5
mx2_error (force)	/cig/othersrc/force/mx2_hword.c	2.13.34.5
mx2_error (rtsw)	/cig/libsrc/librtt/mx2_hword.c	2.13.35.5
mx2_hwcopy (force)	/cig/othersrc/force/mx2_hword.c	2.13.34.6
mx2_hwcopy (rtsw)	/cig/libsrc/librtt/mx2_hword.c	2.13.35.6
mx2_open (force)	/cig/othersrc/force/mx2_hword.c	2.13.34.1
mx2_open (rtsw)	/cig/libsrc/librtt/mx2_hword.c	2.13.35.1
mx2_peek (force)	/cig/othersrc/force/mx2_hword.c	2.13.34.3
mx2_peek (rtsw)	/cig/libsrc/librtt/mx2_hword.c	2.13.35.3
mx2_push (force)	/cig/othersrc/force/mx2_hword.c	2.13.34.2
mx2_push (rtsw)	/cig/libsrc/librtt/mx2_hword.c	2.13.35.2



Function Name	Location	Section
mx2_skip (force)	/cig/othersrc/force/mx2_hword.c	2.13.34.4
mx2_skip (rtsw)	/cig/libsrc/librt/mx2_hword.c	2.13.35.4
mx3_error	/cig/libsrc/libgossip/mx3_hword.c	2.9.17.5
mx3_hwcopy	/cig/libsrc/libgossip/mx3_hword.c	2.9.17.6
mx3_open	/cig/libsrc/libgossip/mx3_hword.c	2.9.17.1
mx3_peek	/cig/libsrc/libgossip/mx3_hword.c	2.9.17.3
mx3_push	/cig/libsrc/libgossip/mx3_hword.c	2.9.17.2
mx3_skip	/cig/libsrc/libgossip/mx3_hword.c	2.9.17.4
mx_error	/cig/libsrc/libball/mx_error.c	2.4.4.1
mx_open	/cig/libsrc/libball/mx_open.c	2.4.4.2
mx_peek	/cig/libsrc/libball/mx_peek.c	2.4.4.3
mx_push	/cig/libsrc/libball/mx_push.c	2.4.4.4
mx_skip	/cig/libsrc/libball/mx_skip.c	2.4.4.5
mx_wcopy	/cig/libsrc/libball/mx_wcopy.c	2.4.4.6
new_pos_orient	/cig/libsrc/libflea/flea_agpt_locations.c	2.7.7.3
next_char	/cig/libsrc/libtoken/lex.c	2.15.1.2
next_line	/cig/libsrc/libtoken/lex.c	2.15.1.5
next_token	/cig/libsrc/libtoken/lex.c	2.15.1.4
nmi_type	/cig/othersrc/force/nmi_type.c	2.13.35
old_mtx_perspective	/cig/libsrc/libvpt/mtx_viewspace.c	2.17.13.3
open_dbase	/cig/libsrc/librt/open_dbase.c	2.13.36.1
open_aed	/cig/libsrc/librt/open_ded.c	2.13.37
open_dr11_interface	/cig/libsrc/libhost/host_dr11_if.c	2.10.1.1
open_enet_interface	/cig/libsrc/libhost/host_enet_if.c	2.10.2.1
open_flea_interface	/cig/libsrc/libhost/host_flea_if.c	2.10.3.1
open_mpv_interface	/cig/libsrc/libhost/host_mpv_if.c	2.10.5.1
open_scsi_interface	/cig/libsrc/libhost/host_scsi_if.c	2.10.6.1
open_socket_interface	/cig/libsrc/libhost/host_socket_if.c	2.10.7.1
OpenDir	/cig/libsrc/libutil/directory.c	2.16.2.1
otherveh_state	/cig/libsrc/librt/otherveh_state.c	2.13.38
outhere	/cig/libsrc/libgossip/replace_mod.c	2.9.18.6
outdisplay	/cig/libsrc/libgossip/dtp_emu.c	2.9.4.3
overlay_setup	/cig/libsrc/libconfig/overlay_setup.c	2.5.7
p_configtree_node	/cig/libsrc/libvpt/tst_ereadconfig.c	2.17.20.2
p_overlay_setup	/cig/libsrc/libvpt/tst_ereadconfig.c	2.17.20.4
p_terrain_feedback	/cig/libsrc/libvpt/tst_ereadconfig.c	2.17.20.6
p_viewport_state	/cig/libsrc/libvpt/tst_ereadconfig.c	2.17.20.3
parse_subsys_file	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.2
pix_mult	/cig/gtbinsrc/rt/mkcal.c	2.1.2.4
poll_ready	/cig/othersrc/force/poll_ready.c	2.13.36
poll_shutdown (ballistics)	/cig/gtbinsrc/bx147_main.c	2.1.1.2
poll_shutdown (rtt)	/cig/gtbinsrc/rt/rtt.c	2.1.3.5
pool_get_off_24	/cig/libsrc/librt/ded_gm_pool.c	2.13.15.2
pool_init	/cig/libsrc/librt/ded_gm_pool.c	2.13.15.1
pop_node	/cig/libsrc/libgendtp/dtp_funcs.c	2.8.2.2
power	/cig/libsrc/libconfig/fill_tree.c	2.5.4.2
ppm_get_data	/cig/libsrc/libbackend/ppm_obj.c	2.3.10.3
ppm_init	/cig/libsrc/libbackend/ppm_obj.c	2.3.10.2
ppm_load	/cig/libsrc/libbackend/ppm_obj.c	2.3.10.4
ppm_setup	/cig/libsrc/libbackend/ppm_obj.c	2.3.10.1
pr_branch	/cig/libsrc/libvpt/tst_treetrace.c	2.17.23.2
pr_matrix	/cig/libsrc/libvpt/tst_treetrace.c	2.17.23.3
pretend_veh	/cig/libsrc/librt/pretend_veh.c	2.13.39
print_msg_1rotation	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_2d_setup	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_3rotations	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_add_traj_table	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_agl	/cig/libsrc/libmsg/print_msg.c	2.12.19.1

Function Name	Location	Section
print_msg_agl_setup	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_ammo_define	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_calibration_image	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_cancel_round	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_cig_ctl	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_cloud_state	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_create_confignode	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_default	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_define_tx_mode	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_delete_traj_table	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_dr11_pkt_size	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_end	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_file_descr	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_file_status	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_file_xfer	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_gun_overlay	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_hit_return	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_hit_return48	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_hprxyzs_matrix	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_laser_return	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_local_terrain	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_lt_piece	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_lt_state	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_miss	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_obscure	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_otherveh_state	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_overlay_setup	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_pass_back	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_pass_on	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_ppm_display_mode	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_ppm_display_offset	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_ppm_pixel_location	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_ppm_pixel_state	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_process_chord	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_process_round	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_process_round48	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_request_laser_range	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_request_point_info	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_return_point_info	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_rot2x1_matrix	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_round_fired	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_rts4x3_matrix	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_scale	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_shot_report	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_shot_report48	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_show_effect	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_sio_close	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_sio_init	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_sio_write	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_staticveh_rem	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_staticveh_state	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_subsys_mode	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_sys_error	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_test_name	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_tf_hdr	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_tf_init_hdr	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_tf_init_pt	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_tf_pt	/cig/libsrc/libmsg/print_msg.c	2.12.19.1

Function Name	Location	Section
print_msg_tf_pt48	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_tf_state	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_tf_vehicle_pos	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_traj_chord	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_traj_entry	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_traj_entry_xfer	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_traj_table_xfer	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_translation	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_view_flags	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_view_magnification	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_viewport_state	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_msg_viewport_update	/cig/libsrc/libmsg/print_msg.c	2.12.19.1
print_pdbase	/cig/libsrc/librt/config_database.c	2.13.12.4
process_2d_setup	/cig/libsrc/libflea/cf_translator.c	2.7.2.14
process_a_msg	/cig/libsrc/librt/simulation.c	2.13.43.2
process_add_traj_table	/cig/libsrc/libflea/cf_translator.c	2.7.2.12
process_agl_setup	/cig/libsrc/libflea/cf_translator.c	2.7.2.10
process_amm0_define	/cig/libsrc/libflea/cf_translator.c	2.7.2.19
process_chord	/cig/libsrc/libflea/encode_routines.c	2.7.4.26
process_cig_ctl	/cig/libsrc/libflea/cf_translator.c	2.7.2.2
process_command	/cig/libsrc/lib2d/proc_cmd.c	2.2.12
process_configtree_node	/cig/libsrc/libflea/cf_translator.c	2.7.2.4
process_define_tx_mode	/cig/libsrc/libflea/cf_translator.c	2.7.2.6
process_dr11_pkt_size	/cig/libsrc/libflea/cf_translator.c	2.7.2.16
process_file_description	/cig/libsrc/libflea/cf_translator.c	2.7.2.3
process_lt_state	/cig/libsrc/libflea/cf_translator.c	2.7.2.15
process_overlay_setup	/cig/libsrc/libflea/cf_translator.c	2.7.2.7
process_ppm_display_mode	/cig/libsrc/libflea/cf_translator.c	2.7.2.20
process_ppm_display_offset	/cig/libsrc/libflea/cf_translator.c	2.7.2.21
process_ppm_pixel_location	/cig/libsrc/libflea/cf_translator.c	2.7.2.22
process_ppm_pixel_state	/cig/libsrc/libflea/cf_translator.c	2.7.2.23
process_round	/cig/libsrc/libflea/encode_routines.c	2.7.4.27
process_sio_close	/cig/libsrc/libflea/cf_translator.c	2.7.2.17
process_sio_init	/cig/libsrc/libflea/cf_translator.c	2.7.2.11
process_tf_init_hdr	/cig/libsrc/libflea/cf_translator.c	2.7.2.8
process_tf_init_pt	/cig/libsrc/libflea/cf_translator.c	2.7.2.9
process_tf_state	/cig/libsrc/libflea/cf_translator.c	2.7.2.18
process_traj_entry	/cig/libsrc/libflea/cf_translator.c	2.7.2.13
process_vflags	/cig/libsrc/libconfig/process_vflags.c	2.5.8
process_viewport_state	/cig/libsrc/libflea/cf_translator.c	2.7.2.5
process_vppos	/cig/libsrc/libconfig/process_vppos.c	2.5.9
prt_mtx	/cig/libsrc/libru/make_bbn.c	2.13.32.1
prtackerr	/cig/libsrc/libmpvideo/bootmpv.c	2.11.2.3
prtmngerr	/cig/libsrc/libmpvideo/bootmpv.c	2.11.2.2
prtstaterr	/cig/libsrc/libmpvideo/bootmpv.c	2.11.2.4
push_node	/cig/libsrc/libgndtp/dtp_funcs.c	2.8.2.1
put_in_hdr	/cig/libsrc/libflea/encode_routines.c	2.7.4.23
r4mat_dump	/cig/libsrc/libvpt/mtx_dump.c	2.17.12.1
r8mat_dump	/cig/libsrc/libvpt/mtx_dump.c	2.17.12.2
rcl_command	/cig/libsrc/libgndtp/rcfuncs.c	2.8.5.11
rcl_component	/cig/libsrc/libgndtp/rcfuncs.c	2.8.5.12
rcl_data	/cig/libsrc/libgndtp/rcfuncs.c	2.8.5.13
rcl_init_adrs	/cig/libsrc/libgndtp/rcfuncs.c	2.8.5.6
rcl_init_stack	/cig/libsrc/libgndtp/rcfuncs.c	2.8.5.1
rcl_lblcmd	/cig/libsrc/libgndtp/rcfuncs.c	2.8.5.10
rcl_patch_adrs	/cig/libsrc/libgndtp/rcfuncs.c	2.8.5.4
rcl_pop	/cig/libsrc/libgndtp/rcfuncs.c	2.8.5.3
rcl_push	/cig/libsrc/libgndtp/rcfuncs.c	2.8.5.2

Function Name	Location	Section
rcl_rtn_adrs	/cig/libsrc/libgendtp/rcfuncs.c	2.8.5.7
rcl_set_cntlbl	/cig/libsrc/libgendtp/rcfuncs.c	2.8.5.9
rcl_set_errptr	/cig/libsrc/libgendtp/rcfuncs.c	2.8.5.5
rcl_set_label	/cig/libsrc/libgendtp/rcfuncs.c	2.8.5.8
rcl_set_modloc	/cig/libsrc/libgossip/gos_model.c	2.9.10.2
rcl_stuff_data	/cig/libsrc/libgendtp/rcfuncs.c	2.8.5.14
read_a_keyword	/cig/libsrc/libflea/cf_translator.c	2.7.2.24
read_evc_control	/cig/libsrc/librtu/sys_control.c	2.13.46.7
ReadDir	/cig/libsrc/libutil/directory.c	2.16.2.2
red_clock	/cig/othersrc/force/forcetask.c	2.18.32.4
remove_comment_lines	/cig/libsrc/libflea/cf_translator.c	2.7.2.26
remove_white_space	/cig/libsrc/libflea/cf_translator.c	2.7.2.25
replace_mod	/cig/libsrc/libgossip/replace_mod.c	2.9.18.1
restart_clock	/cig/othersrc/force/forcetask.c	2.18.32.3
restore_cur	/cig/libsrc/libutil/vt100.c	2.16.7.8
return_aam_ptr	/cig/libsrc/libconfig/aam_manager.c	2.5.1.2
rnd	/cig/libsrc/librtu/clouds.c	2.13.9.8
rotate_x	/cig/libsrc/librtu/make_bbn.c	2.13.32.2
rotate_x_nt	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.2
rotate_y	/cig/libsrc/librtu/make_bbn.c	2.13.32.3
rotate_y_nt	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.3
rotate_z	/cig/libsrc/librtu/make_bbn.c	2.13.32.4
rotate_z_nt	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.4
rowcol_rd_1	/cig/libsrc/librtu/rowcol_rd.c	2.13.40.1
rowcol_rd_1_cleanup	/cig/libsrc/librtu/rowcol_rd.c	2.13.40.6
rowcol_rd_2	/cig/libsrc/librtu/rowcol_rd.c	2.13.40.2
rowcol_rd_2_cleanup	/cig/libsrc/librtu/rowcol_rd.c	2.13.40.7
rowcol_rd_3	/cig/libsrc/librtu/rowcol_rd.c	2.13.40.3
rowcol_rd_3_cleanup	/cig/libsrc/librtu/rowcol_rd.c	2.13.40.8
rowcol_rd_4	/cig/libsrc/librtu/rowcol_rd.c	2.13.40.4
rowcol_rd_4_cleanup	/cig/libsrc/librtu/rowcol_rd.c	2.13.40.9
rt_accept	/cig/libsrc/librtu/rt_mailbox.c	2.13.41.3
rt_pend	/cig/libsrc/librtu/rt_mailbox.c	2.13.41.1
rt_post	/cig/libsrc/librtu/rt_mailbox.c	2.13.41.2
rtu_entry_pt	/cig/libsrc/libmpvideo/bootforce.c	2.11.1.3
rtu_shutdown	/cig/gtbinsrc/rtu/ru.c	2.1.3.4
s_step	/cig/libsrc/libgossip/gossip.c	2.9.15.4
save_cur	/cig/libsrc/libutil/vt100.c	2.16.7.7
scale_mtx	/cig/libsrc/librtu/mkmtx_nt.c	2.13.33.7
scan_number	/cig/libsrc/libtoken/lex.c	2.15.1.3
scratch_flea	/cig/libsrc/libflea/flea.c	2.7.5.9
scroll_reg	/cig/libsrc/libutil/vt100.c	2.16.7.9
se_clock	/cig/othersrc/force/forcetask.c	2.18.32.5
send_ammo_define	/cig/libsrc/libflea/encode_routines.c	2.7.4.32
send_data	/cig/libsrc/libutil/sload.c	2.16.5.3
send_gun_overlay	/cig/libsrc/libflea/encode_routines.c	2.7.4.31
set_3d_lut_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.13
set_color_config_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.15
set_command_2d	/cig/libsrc/libflea/flea_agpt_switches.c	2.7.8.2
set_data_2d_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.14
set_database_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.10
set_ded_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.11
set_entry_pt	/cig/libsrc/libmpvideo/bootforce.c	2.11.1.2
set_esifa_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.15
set_final_lut_name	/cig/libsrc/libtoken/subsys_cfg_parse.c	2.15.2.12
set_leds	/cig/libsrc/librtu/sys_control.c	2.13.46.3
set_lut	/cig/libsrc/librtu/clouds.c	2.13.9.7
set_token_file	/cig/libsrc/libtoken/lex.c	2.15.1.1

Function Name	Location	Section
setup_bit_blt	/cig/libsrc/lib2d/bit_blt.c	2.2.1
setup_comp_start	/cig/libsrc/lib2d/comp.c	2.2.6
setup_define_string	/cig/libsrc/lib2d/string.c	2.2.13
setup_define_window	/cig/libsrc/lib2d/window.c	2.2.19
setup_draw_line	/cig/libsrc/lib2d/draw_line.c	2.2.7
setup_oval_rectangle	/cig/libsrc/lib2d/oval_rect.c	2.2.10
setup_p_terrain_feedback	/cig/libsrc/libvpt/tst_ereadconfig.c	2.17.20.5
setup_poly	/cig/libsrc/lib2d/poly.c	2.2.11
setup_text	/cig/libsrc/lib2d/text.c	2.2.14
sgr	/cig/libsrc/libutil/vt100.c	2.16.7.2
shot_report	/cig/libsrc/libball/shot_report.c	2.4.3.12
show_effect_msg	/cig/libsrc/libmsg/show_effect_msg.c	2.12.20
sim_bal_agl_wanted	/cig/libsrc/librt/bal_routines.c	2.13.4.5
sim_bal_frame_rate	/cig/libsrc/librt/bal_routines.c	2.13.4.3
sim_bal_init	/cig/libsrc/librt/bal_routines.c	2.13.4.1
sim_bal_process_msg	/cig/libsrc/librt/bal_routines.c	2.13.4.6
sim_bal_process_tracer	/cig/libsrc/librt/bal_routines.c	2.13.4.7
sim_bal_req_pt_info	/cig/libsrc/librt/bal_routines.c	2.13.4.4
sim_bal_reset	/cig/libsrc/librt/bal_routines.c	2.13.4.12
sim_bal_round_fired	/cig/libsrc/librt/bal_routines.c	2.13.4.9
sim_bal_start	/cig/libsrc/librt/bal_routines.c	2.13.4.2
sim_bal_static_add	/cig/libsrc/librt/bal_routines.c	2.13.4.10
sim_bal_static_rem	/cig/libsrc/librt/bal_routines.c	2.13.4.11
sim_bal_tf_veh_update	/cig/libsrc/librt/bal_routines.c	2.13.4.13
sim_bal_traj_chord	/cig/libsrc/librt/bal_routines.c	2.13.4.8
simulation	/cig/libsrc/librt/simulation.c	2.13.43.1
single_lite	/cig/libsrc/libgossip/replace_mod.c	2.9.18.2
sio_close	/cig/libsrc/libbio/sio.c	2.14.1.3
sio_init	/cig/libsrc/libbio/sio.c	2.14.1.1
sio_tick	/cig/libsrc/libbio/sio.c	2.14.1.4
sio_write	/cig/libsrc/libbio/sio.c	2.14.1.2
slave_cig_enet_sync	/cig/libsrc/libhost/host_enet_if.c	2.10.2.4
sload	/cig/libsrc/libutil/sload.c	2.16.5.1
spur_int	/cig/othersrc/force/exception.asm	2.18.2.2
staticveh_remove	/cig/libsrc/librt/staticveh_remove.c	2.13.44
staticveh_state	/cig/libsrc/librt/staticveh_state.c	2.13.45
status_mpvideo_print	/cig/libsrc/libmpvideo/mpvideo_print.c	2.11.10.1
stdopen	/cig/libsrc/libutil/stdopen.c	2.16.6
strcat	/cig/othersrc/force/f0_query.c	2.18.22.3
strcpy	/cig/othersrc/force/f0_query.c	2.18.22.2
strlen	/cig/othersrc/force/f0_query.c	2.18.22.4
swallow_token	/cig/libsrc/libtoken/lex.c	2.15.1.6
swap_axis	/cig/libsrc/librt/mkmtx_nt.c	2.13.33.5
sys_control_init	/cig/libsrc/librt/sys_control.c	2.13.46.1
sys_frame_rate	/cig/libsrc/librt/sys_control.c	2.13.46.2
sys_master_sync	/cig/libsrc/librt/sys_control.c	2.13.46.5
sys_slave_sync	/cig/libsrc/librt/sys_control.c	2.13.46.4
syserr	/cig/libsrc/librt/simulation.c	2.13.43.3
system_aam_init	/cig/libsrc/libconfig/aam_manager.c	2.5.1.3
test_commands	/cig/libsrc/libgossip/test_commands.c	2.9.19
test_gsp	/cig/othersrc/force/test_gsp.c	2.13.37
tick	/cig/libsrc/libflea/tick.c	2.7.26.2
tick_init	/cig/libsrc/libflea/tick.c	2.7.26.1
tick_main_menu	/cig/libsrc/libflea/tick.c	2.7.26.4
tick_ppm	/cig/libsrc/libflea/tick_ppm.c	2.7.27.1
tick_ppm_menu	/cig/libsrc/libflea/tick_ppm.c	2.7.27.2
tick_ppm_menu_header	/cig/libsrc/libflea/tick_ppm.c	2.7.27.3
tick_ppm_update_info	/cig/libsrc/libflea/tick_ppm.c	2.7.27.4

Function Name	Location	Section
tick_script	/cig/libsrc/libflea/tick_script.c	2.7.28.1
tick_script_main_menu	/cig/libsrc/libflea/tick_script.c	2.7.28.2
toggle_mpvideo_print	/cig/libsrc/libmpvideo/mpvideo_print.c	2.11.10.2
translate	/cig/libsrc/librtt/mkmtx_nt.c	2.13.33.8
trav_tree	/cig/libsrc/libvpt/trav_tree.c	2.17.17
triple_lite	/cig/libsrc/libgossip/replace_mod.c	2.9.18.4
tst_edebug	/cig/libsrc/libvpt/tst_edebug.c	2.17.18
tst_equery	/cig/libsrc/libvpt/tst_equery.c	2.17.19.1
tst_ereadconfig	/cig/libsrc/libvpt/tst_ereadconfig.c	2.17.20.1
tst_eupdate	/cig/libsrc/libvpt/tst_eupdate.c	2.17.21
tst_tree	/cig/libsrc/libvpt/tst_tree.c	2.17.22.1
tst_treetrace	/cig/libsrc/libvpt/tst_treetrace.c	2.17.23.1
upd_add_static_veh	/cig/libsrc/libflea/encode_routines.c	2.7.4.9
upd_auto_fire	/cig/libsrc/libflea/encode_routines.c	2.7.4.7
upd_chord_fired	/cig/libsrc/libflea/encode_routines.c	2.7.4.6
upd_clouds	/cig/libsrc/libflea/encode_routines.c	2.7.4.14
upd_count_hits_per_min	/cig/libsrc/libflea/encode_routines.c	2.7.4.11
upd_dynamic_matrix	/cig/libsrc/libflea/encode_routines.c	2.7.4.3
upd_flea_vehicles	/cig/libsrc/libflea/encode_routines.c	2.7.4.24
upd_lt_state	/cig/libsrc/libflea/encode_routines.c	2.7.4.30
upd_matrix_values	/cig/libsrc/libflea/encode_routines.c	2.7.4.1
upd_ppm	/cig/libsrc/libflea/flea_ppm_obj.c	2.7.18.1
upd_rem_static_veh	/cig/libsrc/libflea/encode_routines.c	2.7.4.8
upd_req_agl	/cig/libsrc/libflea/encode_routines.c	2.7.4.15
upd_req_lrange	/cig/libsrc/libflea/encode_routines.c	2.7.4.17
upd_req_point	/cig/libsrc/libflea/encode_routines.c	2.7.4.16
upd_rotation_values	/cig/libsrc/libflea/encode_routines.c	2.7.4.2
upd_round_fired	/cig/libsrc/libflea/encode_routines.c	2.7.4.5
upd_send_dynamic	/cig/libsrc/libflea/encode_routines.c	2.7.4.10
upd_send_stop	/cig/libsrc/libflea/encode_routines.c	2.7.4.21
upd_show_eff	/cig/libsrc/libflea/encode_routines.c	2.7.4.13
upd_sio_write	/cig/libsrc/libflea/encode_routines.c	2.7.4.29
upd_subsys_mode	/cig/libsrc/libflea/encode_routines.c	2.7.4.19
upd_view_flags	/cig/libsrc/libflea/encode_routines.c	2.7.4.4
upd_view_mag	/cig/libsrc/libflea/encode_routines.c	2.7.4.18
upd_view_mode	/cig/libsrc/libflea/encode_routines.c	2.7.4.12
upd_viewport_up	/cig/libsrc/libflea/encode_routines.c	2.7.4.20
update_2d	/cig/libsrc/libflea/update_2d.c	2.7.29
update_agpt_2d	/cig/libsrc/libflea/update_agpt_2d.c	2.7.30
update_dyn_demo	/cig/libsrc/libflea/flea_agpt_locations.c	2.7.7.4
update_mag	/cig/libsrc/libflea/flea_agpt_switches.c	2.7.8.3
update_menu_header	/cig/libsrc/libflea/tick.c	2.7.26.6
update_subsys_mode	/cig/libsrc/libflea/flea_agpt_switches.c	2.7.8.4
update_vpt	/cig/libsrc/libflea/flea_agpt_switches.c	2.7.8.5
upstart	/cig/libsrc/librtu/upstart.c	2.13.47.1
upstart_cleanup	/cig/libsrc/librtu/upstart.c	2.13.47.2
vasi_lite	/cig/libsrc/libgossip/replace_mod.c	2.9.18.5
vec_mat_mul	/cig/libsrc/librtt/mkmtx_nt.c	2.13.33.20
vpt_cnode_get	/cig/libsrc/libvpt/cnode_get.c	2.17.3
vpt_cnode_linkvpt	/cig/libsrc/libvpt/linkvpt.c	2.17.10
vpt_cnode_process	/cig/libsrc/libvpt/cnode_process.c	2.17.4
vpt_cnode_qroot	/cig/libsrc/libvpt/cnode_query.c	2.17.5.2
vpt_cnode_query	/cig/libsrc/libvpt/cnode_query.c	2.17.5.1
vpt_cnode_set_bchild	/cig/libsrc/libvpt/cnode_child.c	2.17.2.1
vpt_cnode_set_branch	/cig/libsrc/libvpt/cnode_set.c	2.17.6.1
vpt_cnode_set_matrix	/cig/libsrc/libvpt/cnode_set.c	2.17.6.2
vpt_cnode_set_stdchild	/cig/libsrc/libvpt/cnode_child.c	2.17.2.2
vpt_init_mode_off	/cig/libsrc/libvpt/be_subs.c	2.17.1.8

Function Name	Location	Section
vpt_init_mode_on	/cig/libsrc/libvpt/be_stubs.c	2.17.1.7
vpt_path_get	/cig/libsrc/libvpt/path_init.c	2.17.15.1
vpt_path_init	/cig/libsrc/libvpt/path_init.c	2.17.15.2
vpt_path_process	/cig/libsrc/libvpt/path.c	2.17.14.1
vpt_path_query	/cig/libsrc/libvpt/path_query.c	2.17.16
vpt_path_setup	/cig/libsrc/libvpt/path_init.c	2.17.15.3
vpt_path_update	/cig/libsrc/libvpt/path.c	2.17.14.2
vpt_root_init	/cig/libsrc/libvpt/init_free.c	2.17.9.1
vpt_tree_free	/cig/libsrc/libvpt/init_free.c	2.17.9.3
vpt_tree_init	/cig/libsrc/libvpt/init_free.c	2.17.9.2
vpt_update	/cig/libsrc/libvpt/vpt_update.c	2.17.35
vpt_update_2x1_heading	/cig/libsrc/libvpt/u_rotations.c	2.17.26.1
vpt_update_2x1_pitch	/cig/libsrc/libvpt/u_rotations.c	2.17.26.2
vpt_update_2x1_roll	/cig/libsrc/libvpt/u_rotations.c	2.17.26.3
vpt_update_3x3_matrix	/cig/libsrc/libvpt/u_xfrm.c	2.17.28.2
vpt_update_4x3_matrix	/cig/libsrc/libvpt/u_xfrm.c	2.17.28.1
vpt_update_all	/cig/libsrc/libvpt/u_viewport.c	2.17.27.7
vpt_update_brmask	/cig/libsrc/libvpt/u_brmask.c	2.17.24
vpt_update_fov	/cig/libsrc/libvpt/u_viewport.c	2.17.27.1
vpt_update_fov_lod	/cig/libsrc/libvpt/u_viewport.c	2.17.27.2
vpt_update_heading	/cig/libsrc/libvpt/u_rotations.c	2.17.26.4
vpt_update_hpr	/cig/libsrc/libvpt/u_rotations.c	2.17.26.7
vpt_update_hprxyzs	/cig/libsrc/libvpt/u_xfrm.c	2.17.28.3
vpt_update_lodm	/cig/libsrc/libvpt/u_viewport.c	2.17.27.3
vpt_update_mtx	/cig/libsrc/libvpt/update_mtx.c	2.17.29
vpt_update_near_plane	/cig/libsrc/libvpt/u_viewport.c	2.17.27.4
vpt_update_one_path	/cig/libsrc/libvpt/u_path.c	2.17.25
vpt_update_pitch	/cig/libsrc/libvpt/u_rotations.c	2.17.26.5
vpt_update_rez	/cig/libsrc/libvpt/u_viewport.c	2.17.27.5
vpt_update_roll	/cig/libsrc/libvpt/u_rotations.c	2.17.26.6
vpt_update_rot	/cig/libsrc/libvpt/update_rot.c	2.17.30
vpt_update_scale	/cig/libsrc/libvpt/u_xfrm.c	2.17.28.4
vpt_update_translation	/cig/libsrc/libvpt/u_xfrm.c	2.17.28.5
vpt_update_view_range	/cig/libsrc/libvpt/u_viewport.c	2.17.27.6
vpt_vpt_get	/cig/libsrc/libvpt/vpt_get.c	2.17.31
vpt_vpt_process	/cig/libsrc/libvpt/vpt_process.c	2.17.32
vpt_vpt_query	/cig/libsrc/libvpt/vpt_query.c	2.17.33
vpt_vpt_set	/cig/libsrc/libvpt/vpt_set.c	2.17.34
vpti_change_path_entry_index	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_change_vpt_entry_index	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_get_pathptr_cnode	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_get_ptr_cnode	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_get_ptr_path	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_get_ptr_vpt	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_get_vptptr_cnode	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_increment_path_entry_index	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_increment_vpt_entry_index	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_ptr_cnode	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_ptr_path	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_ptr_vpt	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_state_dcnode	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_state_dlodmult	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_state_dmatrix	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_state_dpath	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_state_dtproc	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_state_dtree_short	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_state_dvptnode	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_state_edtpc	/cig/libsrc/libvpt/globs.c	2.17.8

Function Name	Location	Section
vpti_set_state_efillt	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_set_state_panel_align	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_state_dcnode	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_state_dlodmult	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_state_dmatrix	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_state_dpath	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_state_dtproc	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_state_dtree_short	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_state_dvptnode	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_state_edtpc	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_state_efillt	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_state_panel_align	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_togstate_dcnode	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_togstate_dlodmult	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_togstate_dmatrix	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_togstate_dpath	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_togstate_dtproc	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_togstate_dtree_short	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_togstate_dvptnode	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_togstate_edtpc	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_togstate_efillt	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_togstate_panel_align	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_val_path_entry_index	/cig/libsrc/libvpt/globs.c	2.17.8
vpti_val_vpt_entry_index	/cig/libsrc/libvpt/globs.c	2.17.8
vptq_activept	/cig/libsrc/libvpt/tst_equery.c	2.17.19.6
vptq_brvals	/cig/libsrc/libvpt/tst_equery.c	2.17.19.5
vptq_cnout	/cig/libsrc/libvpt/tst_equery.c	2.17.19.9
vptq_cnptrs	/cig/libsrc/libvpt/tst_equery.c	2.17.19.4
vptq_dynmtx	/cig/libsrc/libvpt/tst_equery.c	2.17.19.8
vptq_grout	/cig/libsrc/libvpt/tst_equery.c	2.17.19.11
vptq_grptrs	/cig/libsrc/libvpt/tst_equery.c	2.17.19.2
vptq_vpout	/cig/libsrc/libvpt/tst_equery.c	2.17.19.10
vptq_vpptrs	/cig/libsrc/libvpt/tst_equery.c	2.17.19.3
vptq_vptbrout	/cig/libsrc/libvpt/tst_equery.c	2.17.19.7
what_node_on_stack	/cig/libsrc/libgendtp/dtp_funcs.c	2.8.2.3
whatdirptr	/cig/libsrc/librt/load_modules.c	2.13.30.4
write_evc_control	/cig/libsrc/librt/sys_control.c	2.13.46.8
write_evc_frame	/cig/libsrc/librt/sys_control.c	2.13.46.9



## E.4 Macro Names Mapped To Source File Locations

The following list shows each macro function used by the real-time software, and identifies the file in which the macro is defined. The third column shows the section number in which the macro is described in this document.

Macro Name	Location	Section
AAM1_TO_AAM2	/cig/include/kludge.h	B.1
AAM2_ADDR	/cig/include/functions.h;	
	/cig/libsrc/libconfig/process_vflags.c	B.2
AAREAD	/cig/include/definitions.h	B.3
ABSVAL	/cig/include/definitions.h, rt_macros.h	B.4
BCOPY	/cig/include/mx_defines.h	B.5
CHECK_FORCE	/cig/libsrc/libgossip/gos_120tx.c, gos_mpv.c	B.6
CHECKROT	/cig/libsrc/libflea/flea_simulate_vehicles.c	B.7
CUBE	/cig/include/rt_macros.h	B.8
DART_ENQUEUE	/cig/include/functions.h	B.9
DED_BOUNDARY	/cig/libsrc/librtt/ded_model_trace.c	B.10
DEGREE_TO_RADIAN	/cig/include/rt_macros.h	B.11
DELETE_ROUND	/cig/include/bx_macros.h	B.12
DELETE_STAT_VEH	/cig/include/bx_macros.h	B.13
DOWNLOAD_DATA	/cig/libsrc/lib2d/cig_link_2d.c	B.14
dtb_bcn	/cig/include/rcinclude.h	B.15
dtb_bcnr	/cig/include/rcinclude.h	B.15
dtb_bcnrs	/cig/include/rcinclude.h	B.15
dtb_bcnz	/cig/include/rcinclude.h	B.15
dtb_bczr	/cig/include/rcinclude.h	B.15
dtb_bczrs	/cig/include/rcinclude.h	B.15
dtb_bczs	/cig/include/rcinclude.h	B.15
dtb_bdgr	/cig/include/rcinclude.h	B.15
dtb_bdgrs	/cig/include/rcinclude.h	B.15
dtb_bdlr	/cig/include/rcinclude.h	B.15
dtb_bdlrs	/cig/include/rcinclude.h	B.15
dtb_bgn	/cig/include/rcinclude.h	B.15
dtb_bgns	/cig/include/rcinclude.h	B.15
dtb_bgz	/cig/include/rcinclude.h	B.15
dtb_bgzs	/cig/include/rcinclude.h	B.15
dtb_blm	/cig/include/rcinclude.h	B.15
dtb_bnz	/cig/include/rcinclude.h	B.15
dtb_bnzs	/cig/include/rcinclude.h	B.15
dtb_bnzrs	/cig/include/rcinclude.h	B.15
dtb_bnzrs	/cig/include/rcinclude.h	B.15
dtb_bpc	/cig/include/rcinclude.h	B.15
dtb_bpcx	/cig/include/rcinclude.h	B.15
dtb_bru	/cig/include/rcinclude.h	B.15
dtb_brur	/cig/include/rcinclude.h	B.15
dtb_brurs	/cig/include/rcinclude.h	B.15
dtb_brus	/cig/include/rcinclude.h	B.15
dtb_brz	/cig/include/rcinclude.h	B.15
dtb_brzr	/cig/include/rcinclude.h	B.15
dtb_brzrs	/cig/include/rcinclude.h	B.15
dtb_brzs	/cig/include/rcinclude.h	B.15
dtb_dot	/cig/include/rcinclude.h	B.15
dtb_elm	/cig/include/rcinclude.h	B.15

852

Macro Name	Location	Section
EXCHANGE_FLEA_DATA	/cig/include/functions.h	B.18
FIND_LM	/cig/include/functions.h, rt_macros.h	B.19
FLTOFX	/cig/include/rt_macros.h	B.20
FREE_LM_CACHE	/cig/include/bx_macros.h	B.21
FXT0881	/cig/include/functions.h, rt_macros.h	B.22
FXT0FL	/cig/include/rt_macros.h	B.23
GET_CHORD_END	/cig/include/bx_macros.h	B.24
GET_DB_POS	/cig/include/bx_macros.h	B.25
GET_LB_FROM_LM	/cig/include/bx_macros.h	B.26
GLOB	/cig/include/ememory_map.h, memory_map.h	B.27
INCR_COMPONENT	/cig/libsrc/libgendtp/rcfuncs.c	B.28
INIT_MTX	/cig/include/functions.h, rt_macros.h	B.29
MAGSQ2D	/cig/include/rt_macros.h	B.30
MAGSQ3D	/cig/include/rt_macros.h	B.31
MALLOC	/cig/include/bx_defines.h	B.32
MAX	/cig/include/rt_macros.h	B.33
MIN	/cig/include/rt_macros.h	B.34
NEW_ROUND	/cig/include/bx_macros.h	B.35
NEW_STAT_VEH	/cig/include/bx_macros.h	B.36
OPEN_FLEA_DATA	/cig/include/functions.h	B.37
OUTPUT_MESSAGE	/cig/libsrc/libflea/cf_translator.c, flea_init_cig_sw.c	B.38
PAGE_FORMAT	/cig/libsrc/libball/bx_probe.c	B.39
	/cig/libsrc/libgossip/gos_bal_query.c	B.40
poly_ab	/cig/include/rcinclude.h	B.40
poly_bvc	/cig/include/rcinclude.h	B.40
poly_efs	/cig/include/rcinclude.h	B.40
poly_efs_r	/cig/include/rcinclude.h	B.40
poly_flu	/cig/include/rcinclude.h	B.40
poly_fsw	/cig/include/rcinclude.h	B.40
poly_gc	/cig/include/rcinclude.h	B.40
poly_inf	/cig/include/rcinclude.h	B.40
poly_lmf	/cig/include/rcinclude.h	B.40
poly_lsc	/cig/include/rcinclude.h	B.40
poly_mmf	/cig/include/rcinclude.h	B.40
poly_pc	/cig/include/rcinclude.h	B.40
poly_poly	/cig/include/rcinclude.h	B.40
poly_rm1	/cig/include/rcinclude.h	B.40
poly_rm2	/cig/include/rcinclude.h	B.40
poly_rm3	/cig/include/rcinclude.h	B.40
poly_rm4	/cig/include/rcinclude.h	B.40
poly_sc	/cig/include/rcinclude.h	B.40
poly_sci	/cig/include/rcinclude.h	B.40
poly_sec	/cig/include/rcinclude.h	B.40
poly_sm1	/cig/include/rcinclude.h	B.40
poly_sm2	/cig/include/rcinclude.h	B.40
poly_sm3	/cig/include/rcinclude.h	B.40
poly_sm4	/cig/include/rcinclude.h	B.40
poly_stamp	/cig/include/rcinclude.h	B.40
poly_tog	/cig/include/rcinclude.h	B.40
poly_vtxe	/cig/include/rcinclude.h	B.40
poly_vtxl	/cig/include/rcinclude.h	B.40
POP_STACK	/cig/libsrc/librt/ded_model_trace.c	B.41
PRINTD4	/cig/libsrc/libgossip/gos_memory.c	B.42
PRINTD8	/cig/libsrc/libgossip/gos_memory.c	B.43
PRINTHEX4	/cig/libsrc/libgossip/gos_memory.c	B.44
PRINTHEX8	/cig/libsrc/libgossip/gos_memory.c	B.45
PUSH_STACK	/cig/libsrc/librt/ded_model_trace.c	B.46

Macro Name	Location	Section
RADIAN_TO_DEGREE	/cig/include/rt_macros.h	B.47
ROOM4LABEL	/cig/libsrc/libgendtp/rcfuncs.c	B.48
ROOMCHECK	/cig/libsrc/libgendtp/rcfuncs.c	B.49
SEND_TF_INFO	/cig/libsrc/librt/bal_routines.c	B.50
SET_OUT_BITS	/cig/include/definitions.h	B.51
SET_OUT_M2BITS	/cig/include/definitions.h	B.52
SET_PPM_DISPLAY_OFFSET	/cig/libsrc/libflea/tick_ppm.c	B.53
SET_PPM_PIXEL_LOCATION	/cig/libsrc/libflea/tick_ppm.c	B.54
SIGN	/cig/include/rt_macros.h	B.55
SQUARE	/cig/include/rt_macros.h	B.56
SYSERR	/cig/include/functions.h	B.57
TODEG	/cig/libsrc/libflea/flea_simulate_vehicles.c	B.58
TORAD	<multiple files; see section B.59>	B.59
toradians	/cig/include/dig_defines.h, vpi_viewport.h; /cig/libsrc/librt/make_bbn.c, mkmtx_nt.c, model_mtx.c	B.60
TRIGGER_FORCE	/cig/include/functions.h	B.61
VME_TO_VMX	/cig/include/kludge.h; /cig/libsrc/librt/cal.c; /cig/libsrc/libmsg/msg_calibration_image.c, msg_effect.c, msg_veh_state.c	B.62
WAIT_FORCE	/cig/include/functions.h	B.63
WAIT_MPVIO	/cig/include/mpvideo.h	B.64
WAIT_MPVREPLY	/cig/libsrc/libmpvideo/bootmpv.c	B.65
XCLOSE	/cig/include/definitions.h	B.66
XLSEEK	/cig/include/definitions.h	B.67
XOPEN	/cig/include/definitions.h	B.68
XREAD	/cig/include/definitions.h	B.69
XWRITE	/cig/include/definitions.h	B.70

**INDEX BY SECTION NUMBER**

2-D Overlay Compiler (/cig/libsrc/lib2d)	2.2
aam_free	2.17.1.9
aam_malloc	2.5.1.1
aam_manager.c	2.5.1
aa_init.c	2.3.1
active_area_init	2.3.1.1
addr_in_pool	2.13.15.3
agpt_init.c	2.13.1
agpt_statistics.c	2.9.1
autopilot.c	2.7.1
b0_aam_centroid.c	2.4.2.1
b0_aam_sw_corner.c	2.4.2.2
b0_add_static_vehicle.c	2.4.2.3
b0_add_traj_table.c	2.4.2.4
b0_bal_config.c	2.4.2.5
b0_bvol_entry.c	2.4.2.6
b0_cancel_round.c	2.4.2.7
b0_cig_frame_rate.c	2.4.2.8
b0_database_info.c	2.4.2.9
b0_delete_static_vehicle.c	2.4.2.10
b0_delete_traj_table.c	2.4.2.11
b0_error_detected.c	2.4.2.12
b0_inapp_message.c	2.4.2.13
b0_lm_read.c	2.4.2.14
b0_model_directory.c	2.4.2.15
b0_model_entry.c	2.4.2.16
b0_new_frame.c	2.4.2.17
b0_print.c	2.4.2.18
b0_process_chord.c	2.4.2.19
b0_process_round.c	2.4.2.20
b0_round_fired.c	2.4.2.21
b0_state_control.c	2.4.2.22
b0_status_request.c	2.4.2.23
b0_tf_init_hdr.c	2.4.2.24
b0_tf_init_pt.c	2.4.2.25
b0_tf_state.c	2.4.2.26
b0_tf_vehicle_pos.c	2.4.2.27
b0_traj_chord.c	2.4.2.28
b0_traj_entry.c	2.4.2.29
b0_undefined_message.c	2.4.2.30
Backend Manager (/cig/libsrc/libbackend)	2.3

backend_branch.c (backend_set_branch)	2.3.2
backend_clear_laser_requests	2.3.5.6
backend_color.c (backend_set_color)	2.3.3
backend_get_object_addr	2.3.5.5
backend_laser.c	2.3.4
backend_laser_request_range	2.3.4.1
backend_man.c	2.3.5
backend_paths.c	2.3.6
backend_reset	2.3.5.2
backend_response	2.3.4.2
backend_send_req	2.3.5.4
backend_setup	2.3.5.1
backend_set_paths	2.3.6.1
backend_sim_init	2.3.5.3
backend_thermal.c (backend_set_thermal)	2.3.7
backend_update_view_paths	2.3.6.2
backend_video.c (backend_set_video)	2.3.8
Ballistics Database Interaction	2.4.3
Ballistics Interface Message Processing	2.4.2
Ballistics Mainline	2.4.1
Ballistics Message Queue Management	2.4.4
Ballistics Processing (/cig/libsrc/libball)	2.4
ball_effect_add.c	2.13.5
bal_buffer_setup	2.13.10.2
bal_get_db_pos.c	2.13.2
bal_get_lm_grid.c	2.13.3
bal_routines.c	2.13.4
bbnctype.c	2.5.2
be_query_buffer_offset	2.17.1.3
be_query_db0	2.17.1.4
be_query_lm_per_lmb_side	2.17.1.6
be_query_num_paths	2.17.1.1
be_qulm	2.17.1.5
be_stubs.c	2.17.1
bit_blt.c (setup_bit_blt)	2.2.1
blank	2.16.7.6
blcopy.c	2.16.1
bootforce	2.11.1.1
bootforce.c	2.11.1
bootmpv.c	2.11.2
buffer_errors.c	2.9.2
bx147_main.c	2.1.1
bx_bvol_int.c	2.4.3.1

bx_chord_intersect.c	2.4.3.2
bx_compute_round.c	2.4.3.3
bx_delete_round	2.4.3.5.2
bx_delete_stat_veh	2.4.3.5.10
bx_dist_sq_pt_line	2.4.3.5.11
bx_find_round_hit	2.4.3.3.5
bx_find_shot_report	2.4.3.3.3
bx_find_vehicle.c	2.4.3.4
bx_free_lm_cache	2.4.3.5.6
bx_functions.c	2.4.3.5
bx_get_chord_end	2.4.3.5.4
bx_get_db_pos	2.4.3.5.3
bx_get_lb_from_lm	2.4.3.5.8
bx_get_lm_data.c	2.4.3.6
bx_get_lm_grid.c	2.4.3.7
bx_guntip_within_db	2.4.3.3.2
bx_init.c	2.4.1.1
bx_model_int.c	2.4.3.8
bx_new_bvol	2.4.3.5.5
bx_new_poly	2.4.3.5.7
bx_new_round	2.4.3.5.1
bx_new_stat_veh	2.4.3.5.9
bx_poly_int.c	2.4.3.9
bx_probe.c	2.4.1.2
bx_reset.c	2.4.1.3
bx_return_miss	2.4.3.3.1
bx_round_tracer_position	2.4.3.3.4
bx_task	2.4.1.4.1
bx_task.c	2.4.1.4
bx_task_cleanup	2.4.1.4.2
bx_tf_copy_msg	2.4.3.10.4
bx_tf_free_tf_pts	2.4.3.10.7
bx_tf_init_pt_cache	2.4.3.10.1
bx_tf_new_tf_pts	2.4.3.10.6
bx_tf_next	2.4.3.10.3
bx_tf_pack.c	2.4.3.10
bx_tf_pts	2.4.3.10.2
bx_tf_pt_data	2.4.3.10.5
bx_trajectory.c	2.4.3.11
cal.c	2.13.6
cancel_round	2.7.4.28
cf_translator.c	2.7.2
check_restart	2.1.3.2

check_sum	2.16.5.4
CIG	1.1.2
CIG Configuration (/cig/libsrc/libconfig)	2.5
CIG-SIM Communication	1.2
cigsimio_buffer_init	2.13.7.6
cigsimio_frame_end	2.13.7.7
cigsimio_get_data	2.13.7.5
cigsimio_msg_in	2.13.7.2
cigsimio_msg_out	2.13.7.3
cigsimio_obj.c	2.13.7
cigsimio_write	2.13.7.4
cig_2d_setup.c	2.2.2
cig_comp_2d.c (compile_2d)	2.2.3
cig_config.c	2.5.3
cig_getm_2d.c (get_msg_2d)	2.2.4
cig_link_2d.c (linkup)	2.2.5
clear	2.3.1.2
clear_line	2.10.4.5
CloseDir	2.16.2.3
close_db.c	2.13.8
clouds.c	2.13.9
cloud_init	2.13.9.1
cloud_mgmt	2.13.9.4
cloud_placement	2.13.9.5
cloud_scud	2.13.9.6
cloud_update	2.13.9.2
cloud_update_model	2.13.9.3
cnode_child.c	2.17.2
cnode_get.c (vpt_cnode_get)	2.17.3
cnode_process.c (vpt_cnode_process)	2.17.4
cnode_query.c	2.17.5
cnode_set.c	2.17.6
comp.c (setup_comp_start)	2.2.6
compare_buffers	2.18.32.2
config_ballistics	2.13.10.1
config_ballistics.c	2.13.10
config_color_table.c	2.13.11
config_database	2.13.12.1
config_database.c	2.13.12
config_translator	2.7.2.1
CSC Descriptions	2
ctoi	2.16.5.7
cup	2.16.7.1



data_type.c	2.18.1
db_mcc_setup.c	2.13.13
debug_initdr.c	2.13.14
ded_add_effect	2.9.3.4
ded_add_model	2.9.3.3
ded_adjust_addr_tables	2.9.3.6
ded_cross_border	2.9.3.9
ded_dtp_trace	2.9.3.12
ded_gm_pool.c	2.13.15
ded_init_mdl_addr	2.9.3.2
ded_load_dtp_code	2.9.3.13
ded_model_end_addr	2.9.3.5
ded_model_offset	2.9.3.14
ded_model_trace.c	2.13.16
ded_object.c	2.9.3
ded_object_debug	2.9.3.16
ded_print_tables	2.9.3.15
ded_process_directory	2.9.3.11
ded_relocate_model	2.9.3.10
ded_setup	2.9.3.1
ded_sub_end	2.9.3.8
ded_uninit	2.9.3.7
derror	2.7.8.7
directory.c	2.16.2
disable_restart	2.1.3.3
Disk Space Requirements	3.1
display	2.9.4.2
dl_man.c (dl_setup)	2.3.9
dl_man.c (dl_setup)	2.13.17
double_bot	2.16.7.4
double_lite	2.9.18.3
double_off	2.16.7.5
double_top	2.16.7.3
download_bvols.c	2.13.18
draw_line.c (setup_draw_line)	2.2.7
DTP Command Generator (/cig/libsrc/libgendtp)	2.8
dtp_compiler.c	2.8.1
dtp_emu	2.9.4.1
dtp_emu.c	2.9.4
dtp_funcs.c	2.8.2
dtp_malloc	2.8.2.5
dtp_malloc_init	2.8.2.6
dtp_travl.c	2.8.3

ntp_trav2.c	2.8.4
dynamic_aam_init	2.5.1.4
dynamic_demo.c	2.7.3
effect_downcount.c	2.13.19
encode_routines.c	2.7.4
errors	2.13.12.2
ESIFA Interface (/cig/libsrc/libesifa)	2.6
esifa_ConfigData	2.6.4.5
esifa_download	2.6.5.5
esifa_fade.c (esifa_set_fade)	2.6.1
esifa_get_object_addr	2.6.4.4
esifa_laser.c	2.6.2
esifa_laser_request_range	2.6.2.1
esifa_laser_return	2.6.2.2
esifa_load.c	2.6.3
esifa_man.c	2.6.4
esifa_query.c	2.6.5
esifa_queue_data	2.6.4.6
esifa_queue_download	2.6.4.7
esifa_read	2.6.5.3
esifa_read_ports	2.6.5.1
esifa_send_queue	2.6.4.8
esifa_send_req	2.6.4.3
esifa_setup	2.6.4.1
esifa_sim_init	2.6.4.2
esifa_special.c (esifa_set_special)	2.6.6
esifa_thermal.c (esifa_set_thermal)	2.6.7
esifa_video.c (esifa_set_video)	2.6.8
esifa_write	2.6.5.4
esifa_write_ports	2.6.5.2
exception.asm	2.18.2
excep_init	2.18.2.1
exchange_dr11_data	2.10.1.2
exchange_dr11_data_sim	2.10.1.3
exchange_enet_data	2.10.2.2
exchange_enet_data_sim	2.10.2.3
exchange_flea_data	2.10.3.2
exchange_mpv_data	2.10.5.2
exchange_mpv_data_sim	2.10.5.3
exchange_scsi_data	2.10.6.2
exchange_scsi_data_sim	2.10.6.3
exchange_socket_data	2.10.7.2
exchange_socket_data_sim	2.10.7.3

extended_ram_available	2.3.1.3
f0_3dlut_download.c	2.18.3
f0_3dlut_switch.c	2.18.4
f0_alllut_switch.c	2.18.5
f0_debug_disable.c	2.18.6
f0_debug_enable.c	2.18.7
f0_final_lut_download.c	2.18.8
f0_final_lut_switch.c	2.18.9
f0_mode_select.c	2.18.10
f0_mpv_init.c	2.18.11
f0_mpv_lut_type_request.c	2.18.12
f0_mpv_peek.c	2.18.13
f0_mpv_poke.c	2.18.14
f0_mpv_poke16.c	2.18.15
f0_mpv_reset.c	2.18.16
f0_mpv_task_control.c	2.18.17
f0_mpv_test.c	2.18.18
f0_mpv_write.c	2.18.19
f0_pass_on.c	2.18.20
f0_pixel_depth_request.c	2.18.21
f0_query	2.18.22.1
f0_query.c	2.18.22
f0_set_display.c	2.18.23
f0_trigger.c	2.18.24
f0_unknown.c	2.18.25
f1_force_init.c	2.18.26
f1_init_jump_table.c	2.18.27
f1_pa_320x240_h	2.18.28.10
f1_pa_320x240_v	2.18.28.6
f1_pa_640x240_h	2.18.28.12
f1_pa_640x240_v	2.18.28.8
f1_pa_640x256_h	2.18.28.11
f1_pa_640x256_v	2.18.28.7
f1_pa_640x480_h	2.18.28.9
f1_pa_640x480_v	2.18.28.5
f1_pa_fb_offset	2.18.28.4
f1_pa_init	2.18.28.1
f1_pa_new_orientation	2.18.28.2
f1_pa_new_resolution	2.18.28.3
f1_pixel_address.c	2.18.28
f1_process_messages.c	2.18.29
f1_setup_environment.c	2.18.30
file_control.c	2.13.20

fill_tree	2.5.4.1
fill_tree.c	2.5.4
find_be_id	2.17.1.2
find_field.c (FindField)	2.16.3
find_gtfn.c (find_fn)	2.16.4
find_pitch_and_roll	2.7.20.2
fire_round	2.7.4.25
flagoff.c	2.17.7
flea	2.7.5.1
flea.c	2.7.5
flea_abs_playback	2.7.19.1
flea_agl_terrain_follow.c	2.7.6
flea_agpt_locations	2.7.7.1
flea_agpt_locations.c	2.7.7
flea_agpt_locations_main_menu	2.7.7.2
flea_agpt_switches	2.7.8.1
flea_agpt_switches.c	2.7.8
flea_agpt_switches_main_menu	2.7.8.6
flea_atp	2.7.9.1
flea_atp.c	2.7.9
flea_atp_main_menu	2.7.9.2
flea_bal_opts	2.7.10.1
flea_bal_opts.c	2.7.10
flea_bal_opts_main_menu	2.7.10.2
flea_calibration_image	2.7.18.2
flea_cleanup	2.7.5.11
flea_db_traverse.c	2.7.11
flea_decode_data.c	2.7.12
flea_demo.c	2.7.13
flea_draw_2d.c	2.7.14
flea_dummy_getchar	2.7.5.4
flea_encode_data.c	2.7.15
flea_error_print	2.7.4.22
flea_getchar	2.7.5.3
flea_graphics_test	2.7.16.1
flea_graphics_test.c	2.7.16
flea_graphics_test_main_menu	2.7.16.2
flea_host_if	2.10.3.3
flea_initialized	2.7.5.8
flea_init_cig_sw.c	2.7.17
flea_IO_mode	2.7.5.5
flea_IO_off	2.7.5.7
flea_IO_on	2.7.5.6

flea_io_task	2.7.5.2
flea_io_task_cleanup	2.7.5.12
flea_ppm_display_mode	2.7.18.3
flea_ppm_display_offset	2.7.18.4
flea_ppm_obj.c	2.7.18
flea_ppm_pixel_location	2.7.18.5
flea_ppm_pixel_state	2.7.18.6
flea_printf	2.7.5.10
flea_prompt	2.7.26.3
flea_script.c	2.7.19
flea_simulate_vehicles	2.7.20.1
flea_simulate_vehicles.c	2.7.20
flea_switches	2.7.21.1
flea_switches.c	2.7.21
flea_switches_main_menu	2.7.21.2
flea_update_pos.c	2.7.22
flea_veh_control	2.7.23.1
flea_veh_control.c	2.7.23
flea_veh_control_main_menu	2.7.23.2
Force Processing (/cig/othersrc/force)	2.18
force.asm	2.18.31
forcetask.c	2.18.32
ftoh	2.9.4.6
func_msg	2.13.12.3
func_msg	2.13.36.2
fxbvtofl	2.13.21.2
fxbvtofl.c	2.13.21
fxbvtofl_020	2.13.21.3
fxbvtofl_dart	2.13.21.1
generic_lm	2.13.22.2
generic_lm.c	2.13.22
getch.c	2.5.6
GetFileName	2.16.2.4
getlmdp	2.13.30.2
getmatrix	2.13.33.10
getside	2.13.30.3
get_3d_lut_name	2.15.2.6
get_binary_data	2.16.5.5
get_char	2.16.5.6
get_color_config_name	2.15.2.9
get_current_token	2.15.1.8
get_database_name	2.15.2.3
get_data_2d_name	2.15.2.7

get_ded_name	2.15.2.4
get_dtp_bank	2.13.46.6
get_esifa_name	2.15.2.8
get_final_lut_name	2.15.2.5
get_lm	2.9.4.9
get_next_message	2.7.19.3
get_next_packet	2.7.19.2
get_next_token	2.15.1.7
get_number_value	2.15.1.9
get_record	2.16.5.2
get_sio_data.c (get_sio_write_data)	2.7.24
get_string_value	2.15.1.10
get_thing.c	2.2.8
get_tx_lut_index.c	2.13.23
get_vehicle_position.c	2.5.5
global_init.c	2.13.24
globs.c (vpti_*)	2.17.8
gossip	2.9.15.1
gossip.c	2.9.15
gossip_cleanup	2.9.15.12
gossip_tick	2.9.15.2
gos_120tx.c	2.9.5
gos_bal_query.c	2.9.6
gos_cigsimio	2.13.7.1
gos_db_query	2.9.7.1
gos_db_query.c	2.9.7
gos_db_query_menu	2.9.7.3
gos_display_db_info	2.9.7.2
gos_dummy_getchar	2.9.15.9
gos_getchar	2.9.15.8
gos_IO_off	2.9.15.11
gos_IO_on	2.9.15.10
gos_locate.c	2.9.8
gos_main_menu	2.9.15.7
gos_memory.c	2.9.9
gos_model	2.9.10.1
gos_model.c	2.9.10
gos_mpv.c	2.9.11
gos_mpvio.c	2.9.12
gos_polys.c	2.9.13
gos_ppm_query	2.3.10.5
gos_ppm_query_menu	2.3.10.6
gos_prompt	2.9.15.6

gos_single_step	2.9.15.5
gos_system.c	2.9.14
gos_timing_printout	2.9.15.3
gsp_io.c	2.18.33
gsp_ioctl_read	2.18.31.4
gsp_ioctl_write	2.18.31.5
gsp_read	2.18.31.3
gsp_reset	2.18.31.6
gsp_write	2.18.31.2
gun_overlays.c	2.13.25
hexdisplay	2.9.4.5
Host Interface Manager (/cig/libsrc/libhost)	2.1
host_disable_all_debug_msgs	2.10.4.3
host_dr11_if.c	2.10.1
host_enable_all_debug_msgs	2.10.4.2
host_enet_if.c	2.10.2
host_flea_if.c	2.10.3
host_if_buffer_init	2.13.24.1
host_if_debug	2.10.4.13
host_if_debug.c	2.10.4
host_if_debug_init	2.10.4.4
host_if_debug_main_menu	2.10.4.10
host_if_debug_menu	2.10.4.11
host_if_debug_tick	2.10.4.12
host_if_disable_debug_msgs	2.10.4.9
host_if_display_enabled_msgs	2.10.4.6
host_if_enable_debug_msgs	2.10.4.8
host_init_packet_sizes	2.13.24.2
host_list_msgs	2.10.4.7
host_mpv_if.c	2.10.5
host_scsi_if.c	2.10.6
host_socket_if.c	2.10.7
How This Document Is Organized	1.5
htof	2.9.4.7
hw_test.c	2.13.26
hxflt	2.9.4.4
id_4x3mtx	2.13.33.6
id_matrix	2.13.32.6
initialize	2.13.42.2
initialize_defaults	2.13.42.1
init_dr11_interface	2.10.1.4
init_dtp_stacks	2.8.2.4
init_enet_interface	2.10.2.5

init_flea_interface	2.10.3.4
init_free.c	2.17.9
init_generic_lm	2.13.22.1
init_mpv_interface	2.10.5.4
init_ports	2.18.31.1
init_print_msg_array	2.12.19.2
init_scsi_interface	2.10.6.4
init_sim.c (init_simulation)	2.13.27
init_socket_interface	2.10.7.4
init_stuff.c	2.2.9
init_subsys_parser	2.15.2.1
lex.c	2.15.1
linkvpt.c (vpt_cnode_linkvpt)	2.17.10
loadmpv.c (mpvideo_load)	2.11.3
load_dbase.c	2.13.28
load_esifa.c	2.13.29
load_modules	2.13.30.1
load_modules.c	2.13.30
local_terrain	2.13.31.1
local_terrain_cleanup	2.13.31.2
loc_ter.c	2.13.31
loc_ter_msg.c	2.12.1
m1_gun_overlay	2.13.25.1
m2_gun_overlay	2.13.25.2
main	2.18.32.1
main	2.1.1.1
main	2.1.3.1
make4x3	2.13.33.21
make_bbn.c	2.13.32
make_bbn_logo.c	2.9.16
make_cal_matrices	2.1.2.2
make_cal_overlay	2.1.2.1
make_cal_patterns	2.1.2.3
make_m1_overlays	2.13.25.3
make_m2_overlays	2.13.25.4
make_p_nt	2.13.33.1
matrix2	2.13.33.11
mat_adjugate	2.13.33.15
mat_copy	2.13.33.13
mat_determinant	2.13.33.18
mat_inverse	2.13.33.19
mat_mult	2.9.4.8
mat_scale	2.13.33.17



mat_transpose	2.13.33.16
mat_vec_mul	2.13.33.14
Memory Requirements	3.2
mem_check	2.17.22.2
menu_header	2.7.26.5
Message Processing (/cig/libsrc/libmsg)	2.12
mkcal.c	2.1.2
mkmtx_nt.c	2.13.33
model_demo.c	2.7.25
model_mtx.c	2.13.34
MPV Interface (/cig/libsrc/libmpvideo)	2.11
mpvideo_boot	2.11.2.1
mpvideo_define_mode	2.11.7.2
mpvideo_get_object_addr	2.11.6.6
mpvideo_laser.c (mpvideo_laser_request_range)	2.11.4
mpvideo_lut.c (mpvideo_set_lut)	2.11.5
mpvideo_man.c	2.11.6
mpvideo_mode.c	2.11.7
mpvideo_pass_back.c	2.11.8
mpvideo_pass_on.c	2.11.9
mpvideo_print.c	2.11.10
mpvideo_query.c (mpvideo_num_paths)	2.11.11
mpvideo_response.c	2.11.12
mpvideo_send_req	2.11.6.5
mpvideo_setup	2.11.6.2
mpvideo_set_mode	2.11.7.1
mpvideo_set_video	2.11.6.1
mpvideo_sim_init	2.11.6.4
mpvideo_stop	2.11.6.3
mpvmsg_from_buf_addr	2.11.1.7
mpvmsg_query_buf_addr	2.11.1.4
mpvmsg_reply_buf_addr	2.11.1.5
mpvmsg_to_buf_addr	2.11.1.6
msg_1rotation	2.12.18.8
msg_3rotations	2.12.18.9
msg_calibration_image.c	2.12.2
msg_cig_ctl.c	2.12.3
msg_dr11.c (msg_dr11_pkt_size)	2.12.4
msg_effect.c (msg_show_effect)	2.12.5
msg_end	2.12.6.1
msg_end.c	2.12.6
msg_hprxyzs_matrix	2.12.18.5
msg_laser.c (msg_laser_request_range)	2.12.7

msg_laser_return.c	2.12.8
msg_lt_state.c (lt_state)	2.12.9
msg_ootherveh_state	2.12.16.1
msg_pass_back.c	2.12.10
msg_pass_on.c	2.12.11
msg_ppm.c	2.12.12
msg_ppm_display_mode	2.12.12.1
msg_ppm_display_offset	2.12.12.2
msg_ppm_pixel_location	2.12.12.3
msg_ppm_pixel_state	2.12.12.4
msg_process_round48.c	2.12.13
msg_rot2x1_matrix	2.12.18.3
msg_rts4x3_matrix	2.12.18.4
msg_scale	2.12.18.7
msg_shell_sort	2.10.4.1
msg_staticveh_rem	2.12.16.3
msg_staticveh_state	2.12.16.2
msg_subsys_mode.c	2.12.14
msg_syserr.c	2.12.15
msg_translation	2.12.18.6
msg_veh_state.c	2.12.16
msg_vflags.c (msg_view_flags)	2.12.17
msg_viewport_update	2.12.18.1
msg_view_magnification	2.12.18.2
msg_vport.c	2.12.18
mtxcpy	2.13.33.12
mtx_concat.c (concat_mtx)	2.17.11
mtx_dump.c	2.17.12
mtx_non_perspective	2.17.13.1
mtx_perspective	2.17.13.2
mtx_viewspace.c	2.17.13
multmatrix	2.13.32.5
mult_4x3mtx	2.13.33.9
mx2_error	2.13.35.5
mx2_error	2.18.34.5
mx2_hwcopy	2.13.35.6
mx2_hwcopy	2.18.34.6
mx2_hword.c	2.13.35
mx2_hword.c	2.18.34
mx2_open	2.13.35.1
mx2_open	2.18.34.1
mx2_peek	2.13.35.3
mx2_peek	2.18.34.3

mx2_push	2.13.35.2
mx2_push	2.18.34.2
mx2_skip	2.13.35.4
mx2_skip	2.18.34.4
mx3_error	2.9.17.5
mx3_hwcopv	2.9.17.6
mx3_hword.c	2.9.17
mx3_open	2.9.17.1
mx3_peek	2.9.17.3
mx3_push	2.9.17.2
mx3_skip	2.9.17.4
mx_error.c	2.4.4.1
mx_open.c	2.4.4.2
mx_peek.c	2.4.4.3
mx_push.c	2.4.4.4
mx_skip.c	2.4.4.5
mx_wcopy.c	2.4.4.6
new_pos_orient	2.7.7.3
next_char	2.15.1.2
next_line	2.15.1.5
next_token	2.15.1.4
nmi_type.c	2.18.35
old_mtx_perspective	2.17.13.3
OpenDir	2.16.2.1
open_dbase	2.13.36.1
open_dbase.c	2.13.36
open_ded.c	2.13.37
open_dr11_interface	2.10.1.1
open_enet_interface	2.10.2.1
open_flea_interface	2.10.3.1
open_mpv_interface	2.10.5.1
open_scsi_interface	2.10.6.1
open_socket_interface	2.10.7.1
otherveh_state.c	2.13.38
outahere	2.9.18.6
outdisplay	2.9.4.3
oval_rect.c (setup_oval_rectangle)	2.2.10
overlay_setup.c	2.5.7
parse_subsys_file	2.15.2.2
path.c	2.17.14
path_init.c	2.17.15
path_query.c (vpt_path_query)	2.17.16
pix_mult	2.1.2.4

poll_ready.c	2.18.36
poll_shutdown	2.1.1.2
poll_shutdown	2.1.3.5
poly.c (setup_poly)	2.2.11
pool_get_off_24	2.13.15.2
pool_init	2.13.15.1
pop_node	2.8.2.2
power	2.5.4.2
ppm_get_data	2.3.10.3
ppm_init	2.3.10.2
ppm_load	2.3.10.4
ppm_obj.c	2.3.10
ppm_setup	2.3.10.1
pretend_vch.c	2.13.39
print_msg.c	2.12.19
print_msg_*	2.12.19.1
print_pdbase	2.13.12.4
process_2d_setup	2.7.2.14
process_add_traj_table	2.7.2.12
process_agl_setup	2.7.2.10
process_ammo_define	2.7.2.19
process_a_msg	2.13.43.2
process_chord	2.7.4.26
process_cig_ctl	2.7.2.2
process_configtree_node	2.7.2.4
process_define_tx_mode	2.7.2.6
process_dr11_pkt_size	2.7.2.16
process_file_description	2.7.2.3
process_lt_state	2.7.2.15
process_overlay_setup	2.7.2.7
process_ppm_display_mode	2.7.2.20
process_ppm_display_offset	2.7.2.21
process_ppm_pixel_location	2.7.2.22
process_ppm_pixel_state	2.7.2.23
process_round	2.7.4.27
process_sio_close	2.7.2.17
process_sio_init	2.7.2.11
process_tf_init_hdr	2.7.2.8
process_tf_init_pt	2.7.2.9
process_tf_state	2.7.2.18
process_traj_entry	2.7.2.13
process_vflags.c	2.5.8
process_viewport_state	2.7.2.5

process_vppos.c	2.5.9
proc_cmd.c (process_command)	2.2.12
prtackerr	2.11.2.3
prtmsgerr	2.11.2.2
prtstaterr	2.11.2.4
prt_mtx	2.13.32.1
pr_branch	2.17.23.2
pr_matrix	2.17.23.3
push_node	2.8.2.1
put_in_hdr	2.7.4.23
p_configtree_node	2.17.20.2
p_overlay_setup	2.17.20.4
p_terrain_feedback	2.17.20.6
p_viewport_state	2.17.20.3
r4mat_dump	2.17.12.1
r8mat_dump	2.17.12.2
rcfuncs.c	2.8.5
rcl_command	2.8.5.11
rcl_component	2.8.5.12
rcl_data	2.8.5.13
rcl_init_adrs	2.8.5.6
rcl_init_stack	2.8.5.1
rcl_lblcmd	2.8.5.10
rcl_patch_adrs	2.8.5.4
rcl_pop	2.8.5.3
rcl_push	2.8.5.2
rcl_rtn_adrs	2.8.5.7
rcl_set_cntlbl	2.8.5.9
rcl_set_errptr	2.8.5.5
rcl_set_label	2.8.5.8
rcl_set_modloc	2.9.10.2
rcl_stuff_data	2.8.5.14
ReadDir	2.16.2.2
read_a_keyword	2.7.2.24
read_evc_control	2.13.46.7
Real-Time Processing (/cig/libsrc/librtt)	2.13
red_clock	2.18.32.4
remove_comment_lines	2.7.2.26
remove_white_space	2.7.2.25
replace_mod	2.9.18.1
replace_mod.c	2.9.18
Resource Utilization	3
restart_clock	2.18.32.3

restore_cur	2.16.7.8
return_aam_ptr	2.5.1.2
rnd	2.13.9.8
rotate_x	2.13.32.2
rotate_x_nt	2.13.33.2
rotate_y	2.13.32.3
rotate_y_nt	2.13.33.3
rotate_z	2.13.32.4
rotate_z_nt	2.13.33.4
rowcol_rd.c	2.13.40
rowcol_rd_1	2.13.40.1
rowcol_rd_1_cleanup	2.13.40.6
rowcol_rd_2	2.13.40.2
rowcol_rd_2_cleanup	2.13.40.7
rowcol_rd_3	2.13.40.3
rowcol_rd_3_cleanup	2.13.40.8
rowcol_rd_4	2.13.40.4
rowcol_rd_4_cleanup	2.13.40.9
rtn_entry_pt	2.11.1.3
RTSW Software Structure	1.3
rtt.c	2.1.3
rtt_init.c	2.13.42
rtt_shutdown	2.1.3.4
rt_accept	2.13.41.3
rt_mailbox.c	2.13.41
rt_pend	2.13.41.1
rt_post	2.13.41.2
save_cur	2.16.7.7
scale_mtx	2.13.33.7
scan_number	2.15.1.3
scratch_flea	2.7.5.9
scroll_reg	2.16.7.9
send_ammo_define	2.7.4.32
send_data	2.16.5.3
send_gun_overlay	2.7.4.31
Serial Device Input/Output (/cig/libsrc/libtio)	2.14
setup_p_terrain_feedback	2.17.20.5
set_3d_lut_name	2.15.2.13
set_color_config_name	2.15.2.16
set_command_2d	2.7.8.2
set_database_name	2.15.2.10
set_data_2d_name	2.15.2.14
set_ded_name	2.15.2.11

set_entry_pt	2.11.1.2
set_esifa_name	2.15.2.15
set_final_lut_name	2.15.2.12
set_leds	2.13.46.3
set_lut	2.13.9.7
set_token_file	2.15.1.1
se_clock	2.18.32.5
sgr	2.16.7.2
shot_report.c	2.4.3.12
show_effect_msg.c	2.12.20
simulation	2.13.43.1
Simulation Host	1.1.1
simulation.c	2.13.43
Simulator	1.1
sim_bal_agl_wanted	2.13.4.5
sim_bal_frame_rate	2.13.4.3
sim_bal_init	2.13.4.1
sim_bal_process_msg	2.13.4.6
sim_bal_process_tracer	2.13.4.7
sim_bal_req_pt_info	2.13.4.4
sim_bal_reset	2.13.4.12
sim_bal_round_fired	2.13.4.9
sim_bal_start	2.13.4.2
sim_bal_static_add	2.13.4.10
sim_bal_static_rem	2.13.4.11
sim_bal_tf_veh_update	2.13.4.13
sim_bal_traj_chord	2.13.4.8
single_lite	2.9.18.2
sio.c	2.14.1
sio_close	2.14.1.3
sio_init	2.14.1.1
sio_tick	2.14.1.4
sio_write	2.14.1.2
slave_cig_enet_sync	2.10.2.4
sload	2.16.5.1
sload.c	2.16.5
spur_int	2.18.2.2
Stand-Alone Host Emulator (/cig/libsrc/libflea)	2.7
staticveh_remove.c	2.13.44
staticveh_state.c	2.13.45
status_mpvideo_print	2.11.10.1
stdopen.c	2.16.6
strcat	2.18.22.3

strcpy	2.18.22.2
string.c (setup_define_string)	2.2.13
strlen	2.18.22.4
subsys_cfg_parse.c	2.15.2
swallow_token	2.15.1.6
swap_axis	2.13.33.5
syserr	2.13.43.3
System Utilities (/cig/libsrc/libutil)	2.16
system_aam_init	2.5.1.3
sys_control.c	2.13.46
sys_control_init	2.13.46.1
sys_frame_rate	2.13.46.2
sys_master_sync	2.13.46.5
sys_slave_sync	2.13.46.4
s_step	2.9.15.4
Task Initialization (/cig/gtbinsrc)	2.1
test_commands.c	2.9.19
test_gsp.c	2.18.37
text.c (setup_text)	2.2.14
tick	2.7.26.2
tick.c	2.7.26
tick_init	2.7.26.1
tick_main_menu	2.7.26.4
tick_ppm	2.7.27.1
tick_ppm.c	2.7.27
tick_ppm_menu	2.7.27.2
tick_ppm_menu_header	2.7.27.3
tick_ppm_update_info	2.7.27.4
tick_script	2.7.28.1
tick_script.c	2.7.28
tick_script_main_menu	2.7.28.2
toggle_mpvideo_print	2.11.10.2
Token Processing (/cig/libsrc/libtoken)	2.15
translate	2.13.33.8
trav_tree.c	2.17.17
triple_lite	2.9.18.4
tst_edebug.c	2.17.18
tst_equery	2.17.19.1
tst_equery.c	2.17.19
tst_ereadconfig	2.17.20.1
tst_ereadconfig.c	2.17.20
tst_eupdate.c	2.17.21
tst_tree	2.17.22.1



tst_tree.c	2.17.22
tst_treetrace	2.17.23.1
tst_treetrace.c	2.17.23
update_2d.c	2.7.29
update_agpt_2d.c	2.7.30
update_dyn_demo	2.7.7.4
update_mag	2.7.8.3
update_menu_header	2.7.26.6
update_mtx.c (vpt_update_mtx)	2.17.29
update_rot.c (vpt_update_rot)	2.17.30
update_subsys_mode	2.7.8.4
update_vpt	2.7.8.5
upd_add_static_veh	2.7.4.9
upd_auto_fire	2.7.4.7
upd_chord_fired	2.7.4.6
upd_clouds	2.7.4.14
upd_count_hits_per_min	2.7.4.11
upd_dynamic_matrix	2.7.4.3
upd_flea_vehicles	2.7.4.24
upd_lt_state	2.7.4.20
upd_matrix_values	2.7.4.1
upd_ppm	2.7.18.1
upd_rem_static_veh	2.7.4.8
upd_req_agl	2.7.4.15
upd_req_lrange	2.7.4.17
upd_req_point	2.7.4.16
upd_rotation_values	2.7.4.2
upd_round_fired	2.7.4.5
upd_send_dynamic	2.7.4.10
upd_send_stop	2.7.4.21
upd_show_eff	2.7.4.13
upd_sio_write	2.7.4.29
upd_subsys_mode	2.7.4.19
upd_viewport_up	2.7.4.20
upd_view_flags	2.7.4.4
upd_view_mag	2.7.4.18
upd_view_mode	2.7.4.12
upstart	2.13.47.1
upstart.c	2.13.47
upstart_cleanup	2.13.47.2
User Interface Mode (/cig/libsrc/libgossip)	2.9
u_brmask.c (vpt_update_brmask)	2.17.24
u_comp_2d.c (compile_2d)	2.2.15

u_getm_2d.c (get_msg_2d)	2.2.16
u_link_2d.c (linkup)	2.2.17
u_main2d.c (main)	2.2.18
u_path.c (vpt_update_one_path)	2.17.25
u_rotations.c	2.17.26
u_viewport.c	2.17.27
u_xfrm.c	2.17.28
vasi_lite	2.9.18.5
vec_mat_mul	2.13.33.20
Viewport Configuration (/cig/libsrc/libvpt)	2.17
vptq_activept	2.17.19.6
vptq_brvals	2.17.19.5
vptq_cnout	2.17.19.9
vptq_cnptrs	2.17.19.4
vptq_dynmtx	2.17.19.8
vptq_grout	2.17.19.11
vptq_grptrs	2.17.19.2
vptq_vpout	2.17.19.10
vptq_vpptrs	2.17.19.3
vptq_vptbrout	2.17.19.7
vpt_cnode_qroot	2.17.5.2
vpt_cnode_query	2.17.5.1
vpt_cnode_set_bchild	2.17.2.1
vpt_cnode_set_branch	2.17.6.1
vpt_cnode_set_matrix	2.17.6.2
vpt_cnode_set_stdchild	2.17.2.2
vpt_get.c (vpt_vpt_get)	2.17.31
vpt_init_mode_off	2.17.1.8
vpt_init_mode_on	2.17.1.7
vpt_path_get	2.17.15.1
vpt_path_init	2.17.15.2
vpt_path_process	2.17.14.1
vpt_path_setup	2.17.15.3
vpt_path_update	2.17.14.2
vpt_process.c (vpt_vpt_process)	2.17.32
vpt_query.c (vpt_vpt_query)	2.17.33
vpt_root_init	2.17.9.1
vpt_set.c (vpt_vpt_set)	2.17.34
vpt_tree_free	2.17.9.3
vpt_tree_init	2.17.9.2
vpt_update.c	2.17.35
vpt_update_2x1_heading	2.17.26.1
vpt_update_2x1_pitch	2.17.26.2

vpt_update_2x1_roll	2.17.26.3
vpt_update_3x3_matrix	2.17.28.2
vpt_update_4x3_matrix	2.17.28.1
vpt_update_all	2.17.27.7
vpt_update_fov	2.17.27.1
vpt_update_fov_lod	2.17.27.2
vpt_update_heading	2.17.26.4
vpt_update_hpr	2.17.26.7
vpt_update_hprxyzs	2.17.28.3
vpt_update_lodm	2.17.27.3
vpt_update_near_plane	2.17.27.4
vpt_update_pitch	2.17.26.5
vpt_update_rez	2.17.27.5
vpt_update_roll	2.17.26.6
vpt_update_scale	2.17.28.4
vpt_update_translation	2.17.28.5
vpt_update_view_range	2.17.27.6
vt100.c	2.16.7
whatdirptr	2.13.30.4
what_node_on_stack	2.8.2.3
window.c (setup_define_window)	2.2.19
write_evc_control	2.13.46.8
write_evc_frame	2.13.46.9
_copy_reconfigurable_viewports_section	2.12.6.13
_database_disable	2.12.6.10
_display_lights	2.12.6.3
_downcount_effects	2.12.6.2
_handle_point_lights	2.12.6.11
_handle_request_local_terrain	2.12.6.9
_move_load_module_stp_to_quad_buffer	2.12.6.4
_pend_on_frame_interrupt	2.12.6.6
_process_agl	2.12.6.7
_reset_model_pointers	2.12.6.12
_rowcol_rd	2.13.40.5
_rowcol_rd_cleanup	2.13.40.10
_set_up_for_next_frame	2.12.6.8
_update_second_active_area_memory	2.12.6.5